ADVANCED TOPICS IN SCIENCE AND TECHNOLOGY IN CHINA

Yueting Zhuang
Yunhe Pan
Jun Xiao

A Modern Approach to Intelligent Animation

Theory and Practice

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

Springer

ADVANCED TOPICS
IN SCIENCE AND TECHNOLOGY IN CHINA

Yueting Zhuang
Yunhe Pan
Jun Xiao

A Modern Approach to Intelligent Animation

Theory and Practice

With 200 figures

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

Springer

AUTHORS:

Prof. Yueting Zhuang,
College of Computer Science,
Zhejiang University,
310027, Hangzhou, China
E-mail: yzhuang@cs. zju. edu. cn

Prof. Yunhe Pan,
College of Computer Science,
Zhejiang University,
310027, Hangzhou, China

Dr. Jun Xiao
College of Computer Science,
Zhejiang University,
310027, Hangzhou, China
E-mail: junx@cs. zju. edu. cn

ISBN 978-7-308-05454-6 **Zhejiang University Press, Hangzhou**
ISBN 978-3-540-73759-9 **Springer Berlin Heidelberg New York**
e-ISBN 978-3-540-73760-5 **Springer Berlin Heidelberg New York**

Series ISSN 1995-6819 Advanced topics in science and technology in China
Series e-ISSN 1995-6827 Advanced topics in science and technology in China

Library of Congress Control Number: 2007937706

Cover design: Joe Piliero, Springer Science + Business Media LLC, New York
Printed on acid-free paper

Preface

With the explosive evolution of computer hardware and software technology, computer animation has been no longer a mystery in our daily life. The special effects, photorealistic scenes or vivid characters in movies, video games and commercials are all owed to the modern computer animation techniques. In recent years, computer animation is always a very hot topic in the research field of computer graphics. Researchers, engineers and artists all around the world are seeking ways to broaden computer animation's applications, to make animator's work more convenient and to make the world more colorful by computer animation.

By the title of this book, A Modern Approach to Intelligent Animation: Theory and Practice, the authors intend to give readers two impressions: first, intelligent techniques are the tide of modern computer animation; second, the marriage of theory and practice is quite crucial in computer animation, because neither theory nor innocent effort can produce perfect animation works alone.

In recent animation industry, some widely used commercial tools, such as Maya, 3D Max, Softimage and Motion Builder, have produced abundant wonderful results. Yet to some extent, producing animation with these tools still needs many interactions among experienced animators and lacks intelligent assistance, leading to low efficiency and high cost. On the other hand, many new intelligent theories and techniques that could be applied in the animation production are constantly reported in the top international graphics conferences (i. e. SIGGRAPH, EuroGraphics, CGI, SCA, CASA, etc.). But most of these novel theories and techniques have not been transferred into industrial applications. So from this point of view, the theories of animation are apart from the practices.

Motivated by these phenomena and inspired by the rich experiences and research results in computer animation, the authors think that it is essential to write a book that combines both the theories and the practices in intelligent computer animation. In this book, the authors introduce some original theories and algorithms of intelligent computer animation. Moreover, these abstract theories and algorithms are demonstrated by some prototype systems developed by the authors, which might help readers not only understand but also practice these theories and algorithms as well.

The authors plan to achieve two goals by this book: first, for engineers, this book could be a guidance for developing intelligent computer animation applications; second, this book could be a helpful reference for researchers and a shortcut for new arrivals to the research field of intelligent computer animation.

This book contains 8 chapters. Each chapter is designed to introduce an independent topic and stands alone, which is convenient for readers to select the chapter they are interested in. Chapter 1 introduces the preliminaries of computer animation techniques and the background of intelligent computer animation. Chapters 2, 3 and 4 discuss in details video-based human motion capture techniques, with each chapter focusing on a different variety of methods. Video-based facial animation techniques are discussed in Chapter 5. Intelligent motion data preprocessing and management techniques are presented in Chapter 6. In Chapter 7, intelligent motion data reusing methods are introduced. Chapter 8 describes intelligent approaches for character animation.

This book presents our researches in recent 10 years in the area of intelligent animation. Our sincere thanks go to the National Natural Science Foundation of China who supported our research under Grants No. 60525108, No. 60533090.

The authors are extremely thankful to the following gifted colleagues and students who have contributed significantly to the publication of this book. Prof. Chun Chen, Executive Dean of College of Computer Science, Zhejiang University, gave us numerous valuable suggestions and supports during the whole process. Prof. Fei Wu, the key member of our group, has made great contributions in this research, owing to his creative idea and his high sense of responsibility. Dr. Xiaoming Liu, before his PhD study at CMU, did a lot of excellent work in video-based tracking as his master degree work at ZJU. Dr. Zhongxiang Luo did many talented work in tracking and animation during his PhD study at ZJU. We also wish to thank Dr. Qiang Zhu, Dr. Cheng Chen, Dr. Jian Zhang, Dr. Feng Liu, Dr. Yushun Wang, Dr. Jun Yu, Xin Zhou and etc. We have got so much help and support, consequently it is impossible for us to list all of them. Without their contributions, this book could not have been written.

Last, but not least, our thanks go to the Zhejiang University Press, Dr. Qiang Fu, the head of the press, Ms. Xiaojia Chen, Dr. Xinping Qian, Mr. Jun Feng and so many editors. We are especially thankful to those reviewers who did a lot of work for this book. Their responsibility and encouragement help the birth of this book.

Contents

1

Introduction

In recent years, computer animation has been a highly active research topic and is widely applied in various fields such as movie special effects, advertisements, cartoon, computer games and computer simulation, etc. From a traditional perspective of view, researchers categorize computer animation techniques into the field of computer graphics; however, with the fast development of computer animation techniques and enrichment of animation producing facilities, computer animation is no longer restricted to traditional computer graphics category but rather refers to many research areas, such as image processing, digital signal processing, machine vision and artificial intelligence, etc. , to become an interdisciplinary subject.

The subject of computer animation can date back to the 1960s. As the fast growing of computer hardware and theory of computer graphics, computer animation has penetrated to every aspect of life, including television, movie, education, industry, science, etc. Computer animation techniques can be categorized into two domains: model animation growing with the traditional computer animation, and motion capture animation rising in recent years. In this chapter, we will introduce some commonly used computer animation techniques briefly.

1.1 Traditional Computer Animation Techniques

1.1.1 Key-frame Animation

Key-frame animation can be categorized into key-frame interpolation and spline driven animation. The problem solved in these two methods is to compute the position in a certain frame given by the trajectory of a moving object. The motion trajectory is often represented by parametric splines. Interpolation with equidistance will arouse the problem of ill-proportioned motion. In order to obtain the well-proportioned motion, we have to parameterize the spline. Assume that the length of arch is $s=A(\mu)$, where μ is parameter. In order to calculate the value of parameter for the given arch, we need to solve the function $A^{-1}(s)$, which has no analytic solution. Accordingly we solve it numerically, usually by dichotomy algorithm. In the above solution process, we need to compute the length of an arch with a certain parameter value using Simpson method. Guenter and Parent[1] proposed a Gaussian numerical integration method, which used Newton-Raphson iteration method instead of dichotomy method and stored the pre-computed parameter and arch length values to accelerate computation, to replace Simpson method for computing length of arch in order to build correspondence between arch length and parameters effectively. Watt[2] employed forward difference plus search table to speed up interaction. In situation of non-high precision, this method is very effective.

Key-framing can be regarded as a parameter interpolation problem. For the time control in the interpolation, Steketee and Badler[3] used positional splines and motion splines to control motion parameters, where the positional spline is a function of position with respect to key-frames and the motion spline is a function of key-frames with time. Kochanek and Bartels[4] adopted cubic interpolation spline suitable for key-framing system, where they segmented the tangent vector into incoming vector and outgoing vector and introduced three parameters: tensor $\boldsymbol{t}$, continuity $\boldsymbol{c}$ and offset $\boldsymbol{b}$ to control the spline. The tensor $\boldsymbol{t}$ is used to control the curve degree, continuity $\boldsymbol{c}$ for continuity control of key-frames and offset $\boldsymbol{b}$ for overshooting or undershooting control of key-frames. This method allows animators to adjust the movement of object without adjusting the key-frames.

In the key-framing animation, the interpolation of key-frame parameters is usually independent. In this way, the connections between parameters will result in unnatural motion. Brotman and Netravali[5] employed a method of differential functions from classical mechanics to describe the motion constraints from key-frames. Extra forces will be added in the control to satisfy these constraints. The smooth and natural motion could be obtained by energy control in minimizing the coarseness in trajectory. This method is applied for optimal control in inter-connected key-frames, where the parameters are position, face direction, linear and angular velocity, etc.

In the key-frame interpolation for facing directions of moving objects, Eulerian angle is often used. The rotation matrices of Eulerian angle is unexchangeable, therefore the rotations need to be performed in a certain sequence. Besides, equal variation of Eulerian angle will not lead to equal rotation variation, which leads to the asymmetry of rotation. Furthermore, Eulerian angle may result in lose of freedoms. Aiming at the limitations of

Eulerian angle, researchers introduce the concept of quaternion in the following papers.

Shoemake[6] first introduced quaternion to animation, and proposed to apply Béizer spline on unit quaternion to perform interpolation. This method can be used for local control but is opaque to users. It is difficult for users to control vertices by adjusting quaternion.

Duff[7] used B-spline instead of Bézier spline for smoothing rotation. This method can achieve C^2 continuity, but the generated curves do not pass the control vertices and are hard to control.

In order to solve these above problems, Pletincks[8] implement interpolation by four-point normal curves in space of quaternion to control vertices. This method has advantage in solving the control problem of quaternion.

Barr, et al.[9] used a quaternion with an Eulerian angle constraint to smoothly interpolate the facing directions, which allows users to apply constraint on the end points of trajectory. First they transformed the rotation angle to the quaternion, and then minimized the tangent acceleration of quaternion path in non-Euclidean space, and finally used finite difference and optimization algorithms to solve the energy equations.

The main idea of key-framing is to interpolate the transition frames between several given key-frames. Key-framing can be categorized into two classes such as key-frame interpolation and spline driven animation. In the traditional computer 2D cartoon and 3D animation production, key-framing techniques are widely used. Commercial animation software such as 3D MAX, Maya, and Motion Builder, etc. , are all facilitated with key-framing to generate animation sequence.

The key-framing can be effective in scenarios which requiring less accuracy. However, considering the demand of reality and efficiency in modern animation production, key-frame has several deficiencies as below.

- Low efficiency: the key-frames need to be handcrafted by animators to enable computer interpolation.
- High experience requirement: The choice of key-frames will to a large extent affect the final production and efficiency, which requires abundant experience of animators.
- Low reality: In the object movement especially character animation, animation generated by hand-adjusting key-framing are less realistic in behavior or facial expression.

1.1.2 Articulated Animation

In 3D computer animation, character animation is one of the most challenging topics because character body has more than 200 freedoms and very complicated motion, character body is of irregular shapes, muscles are deformed with motion and there are abundant facial expressions. Besides, inconsistent motion will be easily recognized due to the human familiarity with their own motions.

Articulation animation is often represented by articulation model. Articulation model is n-tuple tree, where each inner node is an articulation with a freedom of translation or rotation. For human body, there is only rotation articulation. The motion of articulation is controlled by kinematic or dynamic methods. Forward kinematics sets key-frames by rotation angle to obtain the positions of connected limbs. Inverse kinematics calculates the position of intermediate joints by assigning the position of end joints. Their basic idea is shown below.

Kinematic Methods

Kinematic methods include forward kinematics and inverse kinematics as shown in Fig. 1. 1. Forward kinematics sets key-frames by rotation angle to obtain the positions of connected limbs. In the animation system Pinocchico by Maiocchi and Pernici [10], the authors segmented the real human motion information and stored them in database, and then used animation script to guide the motion of character. Inverse kinematics calculates the position of intermediate joints by assigning the position of end joints, which can be very complicated as the growth of articulation complexity. The cost of solution will be higher and higher, where numerical methods become a feasible solution.

Korein and Badler [11] proposed an intuitive method using hierarchical working space for each articulation segment, which tries to minimize the displacements of articulation positions. The drawback is that users cannot control the obtained results. For a complex articulation structure, the results may not be natural.

Girard and Maciejewski [12,13] used inverse kinematics to generate articulation animation. In this method, users assign the coordinates of feet. The rotation angle from feet to hip will be obtained by solving pseudo inverse Jacobian matrix. This is one of the best methods for generating realistic articulation movement.

An advantage of kinematic methods is that we can set constraints for some key positions of an articulation. For example, when a performer bends one's knees, the feet can be restricted on the floor to lean one's body.

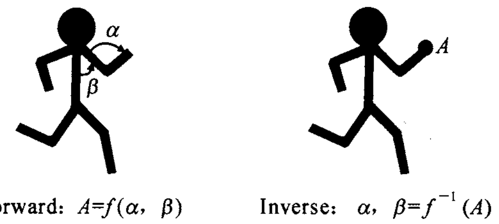

α
β
A
Forward: $A=f(\alpha, \beta)$
Inverse: $\alpha, \beta=f^{-1}(A)$

Fig. 1. 1 Forward kinematics (left) and inverse kinematics (right)

When applying kinematics, we can set a constraint from dynamics. Kinematics and dynamics combination allow animators to be flexible. Isaacs and Cohen [14] proposed a kinematics and dynamics combined system DYNAMO, which has three original characteristics: (1) embed traditional key-framing system into dynamic analysis as a kinematic constraint; (2) able to represent the reaction behavior by surrounding environment through a behavior function; (3) generate specific motion forces through inverse kinematics.

Boulic, et al [15] combined forward and inverse kinematics together to edit articulation motion, where animators could revise the motion towards an object based on user interaction. The key idea of this method is to plug the desired spatial motion into the inverse kinematic mechanism.

Philips and Badler [16] proposed a bipod animal motion control system using interactive kinematic constraints, which could both grasp the motion characteristics and provide balance and stable results.

In summary, inverse kinematic methods are simpler than forward kinematic method, but with high computation cost. Articulation animation requires not only tedious labor of animators but also high computation power. Because it is only a simulation of real human motion, the obtained motion is usually not realistic enough. Therefore some researchers proposed dynamic methods to control articulation motion.

Dynamic Methods

Compared with kinematic methods, dynamic methods can produce more complicated and realistic motions and need less specification of parameters. However, dynamic methods are of high computation cost and hard to control.

Wilhelms and Barsky [17-19] proposed a matrix method applying a generalized force of a freedom only to consider the actual freedom of motion. Therefore the reduced articulation constraint needs not to be adopted in separate equations. The drawback of this method is that the matrix is not sparse, which requires high computation cost to obtain acceleration. Therefore this method is not often used.

Armstrong and Green [20,21] adopted a recursive method from graphic simulation to avoid the reconstruction of matrix. The time complexity of this recursive method is linear with respect to the number of freedoms, which is fast and stable.

Besides the computation complexity, another important problem in dynamic methods is motion control. If there are no effective control methods, users have to provide control instructions such as force or moments, which are almost impossible. Therefore, it is necessary to provide high level control and coordination facilities. High level control depends on lower level control, such as response from collision, impact of friction and damp, articulation constraints to avoid structural artificiality, avoiding unnatural motion, position constraint of a certain joint, etc. A method satisfying the above requirements is pre-processing method, which transforms the required constraints and control to proper forces and moments and plugs into dynamic systems. Witkin, et al [22] proposed a temporal and spatial constraint method by minimizing an object function, which can be solved by conjugate gradient method.

An advantage of dynamics is to simulate the reciprocity between objects, which refers to two problems: occurrence time and response after interaction. Moore and Wilhelms [23] proposed an equation set to describe the momentum conservation and used analytical method to solve the new position and velocity after collision. The collision detection and response increase the reality of simulation but also the computation cost.

Motion Control

The early study of motion control can be found in Zeltzer's work [24]. In his object oriented system, some human motion like walking and jumping was implemented. However, when computing the rotation angle of an articulation, he used kinematics and interpolation between test data, therefore could not realize motion control like alternation of velocity and step size.

Bruderlin and Calvert [25] proposed a mixture method for human walking, which combined object-based and kinematic control techniques. They integrated the cycle motion into a hierarchical control procedure, where the required motion could be specified on the upper level conveniently (such as walking at speed v), and split into low level small tasks, which could be solved by dynamic models. In their experimental system KLAW, after users specify some parameters such as speed, step length and step frequency, the large scale human walking could be generated almost in real-time. In the procedural control method [26], cubic and linear interpolations replace the original dynamic method, keeping nearly all the reality. Therefore, animators could almost control human motion interactively in real-time.

In the real-time human walking model proposed by Boulic and Thalmann [27], the walking model comes from experimental data divided into two levels. The first level was to generate spatio-temporal parameters and the second level used the parameterized trajectory to generate the spatial position of human articulations. Their kinematic method includes the dynamic characteristics of human walking.

Raibert and Hodgins [28] proposed a dynamic control method for motion with legs, where animals could move at different speed and step (running, jogging, galloping and hopping).

McKenna and Zelter [29] proposed a forward kinematic simulation algorithm, whose complexity varied linearly according to the number of articulations and a motion coordination strategy for hexapod animals.

Specifying the motion of articulated animals to achieve an object with the reality of physical laws is one of the purposes of animators. Witkin and kass [30] proposed a new spatio-temporal constraint method, where animators specified content of motion, physical structure of character and physical resources for characters to complete the motion. Based on the description below, plus the Newton law, an optimization problem with constraints is solved to obtain the motion complying with physical laws. The realistic motion generated by this method conforms to some principles of traditional animation. The spatio-temporal constraint leads to a nonlinear problem with constraints, which often has no single result. One of the solutions is to reduce possible trajectories by cubic B-spline basis functions and use constraint optimization to solve the coefficients of B-splines. However, the general solution of these nonlinear optimization problems is often unknown. Therefore Cohen [31] used a symbolic and numerical comprehensive method to realize interactive control, where users could guide the iterative numerical process to convergence. However, as the number of articulations and time complexity increase, the computation cost is still high. This complexity comes from the choice of generalized freedom finite basis. Liu, et al. [32] used wavelet basis to represent the generalized freedom function with time, which had the advantage that we could increase the motion detail as needed to reduce the discrete variables to minimum to get a faster convergence.

In computer animation, it is difficult to build interesting and realistic virtual objects and to keep their control. We need to compromise between complexity and effectiveness of control. For a fixed articulation structure, commonly used optimization methods come from automatic dynamical control systems, e. g. Ngo and Marks'[33] stimulation-reaction algorithm, Van de Panne's sensor network algorithm. These algorithms successfully generated the 2D rigid model motion. For non-stationary 3D objects, Sims [34] obtained autonomous 3D virtual animals, which did not require tedious designing instructions. The morphology and nurse system to control muscles are generated automatically by algorithms.

1.1.3 Facial Expression Animation

Since the pioneering work of 3D human facial animation in 1972, many researches have been done. However, due to the complex structure of facial anatomy and subtle non-rigid motions which are hard to be modeled mathematically, and also due to the human familiarity with facial appearance, this research topic is very difficult. Currently realistic 3D facial animation can be categorized into the following classes.

Interpolation method is the basis for the earliest facial animation. First animators designed several key-frames of facial expressions. Then linear interpolation was used to create transitions between these key-frames. This method mainly depends on the capability of animators and is time-consuming.

Free-form deformation method is to transform facial animation to a surface deformation problem. Several control vertices form a bounding volume, whose motion directly drives facial mesh deformation. This method has less input constraints but requires manual segmentation of face, which is difficult and tedious for ordinary users. Besides, free form deformation does not consider the topology of facial mesh, which leads to some distortions.

Physical model based method is to approximate the anatomical structures of the face, i. e. skull, muscles and skin. The animation from physical models reflects the underlying tissue stresses. Due to the complex topology of human faces, it requires tedious tuning to animate a new face.

Example based method uses motion vectors to estimate the deformation parameters or blending shape coefficients. Predefined morph targets are then blended with respect to the estimated parameters or coefficients. These approaches are attractive for their stable and accurate results. Nevertheless, the processes of handcrafting of morph targets themselves are expensive and time-consuming.

Expression cloning method is to map the displacements of vertices on source model to target meshes. The motion displacements are scaled and rotated with respect to the local detail geometry of source and target meshes for pre-processing. This method works well with the mapping between dense source models and similar target meshes. However, animation by mapping sparse facial motion data to static face models remains unsettled.

Real-time 3D capture method captures both 3D geometry and texture information for facial animations. Photorealistic facial expressions can be reconstructed by deforming the underlying face model with respect to the captured data. The generated head highly resembles the performer. Therefore it is not suitable for animating a given static face model.

1.2 Motion Capture Based Animation Techniques

1.2.1 Definition of Motion Capture

Motion capture is one of the hottest research directions in computer animation field. It includes measuring position and direction in physical space and recording by computers. It captures human body, facial expressions, camera and lighting positions and other elements in the scene. Once the information is recorded in computers, animators can use it as material to generate character animation and virtual scene.

In the synthesized animation, animators usually need to control the path and properties of scene elements by visually simulation. Motion capture animation can synthesize the motion by specifying object path, event time and property control. In the computer generated scene, pure motion capture animation uses the position and direction of real objects to generate movements for synthesized objects. However, due to the constraints by geometric volume matching, precision of motion capture data and requirements of creativity, there is no pure motion capture animation. Even some strict behavior driven systems such as behavior animation have some pre-programming techniques.

1.2.2 Introduction of Motion Capture Techniques

The use of motion capture for computer character animation is relatively new, having begun in the late 1970's, and only now beginning to become widespread. Sturman[35] introduced the brief history of motion capture for computer character animation in detail. Animation by motion capture is mapping captured motion to computer generated virtual objects. Usually, the object of motion capture is the motion of human and animals, where special markers are attached on the joints of objects and tracked position and direction by special hardware.

A motion capture system often includes perception and processing, whose complexity is co-related and compromised. Perception includes active one and passive one. The active perception based motion capture system, applied in fields like sports performance analysis and human computer interface, is simple and widely used, but has high requirement of environment control. The passive perception based on natural signal sources, such as natural light or electro-magnetic waves, does not require wires. The passive perception is applied for intelligent surveillance and human machine interface control. The choice between two perception techniques depends on compromise in complexity.

Motion capture systems usually make some assumptions on capture conditions, such as motion restrictions of body or camera and appearance restrictions of environment and body. Some commercial motion capture systems ranging from simple mechanical system to complicated optical system can be divided into following domains.

Mechanical system consists of potentiometer to measure the position and direction of body articulations. The drawback is that the obtained reality largely depends on the capability and perseverance of animators.

Magnetic system may be one of the most popular systems nowadays. The 3D positions and the relative angles can be acquired by measuring the electro-magnetic field using magnetic sensors. The disadvantages include: (1) sensitive to metal materials in the capture region; (2) restrictions by wired sensors; (3) low sampling rate hardly satisfies the requirements in sport motion analysis. Besides, this system is very expensive.

Optical system attaches reflective markers on human body, uses special cameras to capture video of high contrast, and extracts human motion from videos. It usually requires 4 to 12 cameras. It can provide high motion data sampling rates. However, the drawback is that the system is required to track visual features and 3D reconstruction using computer vision techniques and is highly expensive. As shown in Fig. 1. 2.

Many motion capture systems have been developed so far. Table 1. 1 shows some classical historical moments and representative systems.

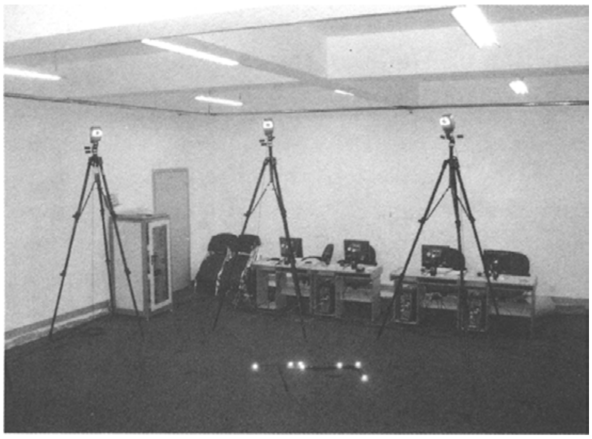


Fig. 1. 2 New motion analysis ™ motion capture system

1. 2. 3 Summarization

Nowadays, various motion capture systems are used to capture the realistic human behavior and expression data used in the data driven animation production. We will describe the characteristics and shortages in existing motion capture systems in the following.

Electro-mechanical system places potentiometer on the articulations of body and uses cables to obtain the 3D positions of them directly. In a sense, it resembles the traditional stop-motion technique and is a natural transition of motion capture techniques. However, it largely depends on the capabilities and endurance of animators.

Magnetic system may be one of the most popular systems nowadays. Both the 3D positions and the relative angles can be acquired by magnetic sensors. The main advantage is that it is efficient, i. e. real-time. It also has some disadvantages, such as sensitive to metal materials in the capture region, restricted by wired sensors and low sampling rates.

Table 1.1 The development of motion capture systems

Age	Historical Moments & Systems	Description
1980—1983	Simon Fraser University—Goniometers [36]	Attached potentiometers to a body and used the output to drive computer animated figures for choreographic studies and clinical assessment of movement abnormalities
1982—1983	Graphical Marionette [37]	Used two cameras with special photo detectors to obtain a 3D world coordinate for LED markers
1988	deGraf/Wahrman—Mike, the Talking Head [38]	Mike, was driven by a specially built controller that allowed a single puppeteer to control many parameters of the character's face
1988	Pacific Data Images—Waldo C. Graphic [39,40]	Control the position and mouth movements of a low resolution claracter in real-time by hooking a custom eight degrees of freedom input device through the standard SGI dial box
1989	Kleiser-Walczak—Dozo [41]	Optical motion analysis method was used to locate the position of reflective tape placed on the body using several cameras
1991	Videosystem—Mat the Ghost [42]	A real-time character animation system
1992	SimGraphics—Mario	A facial tracking system using mechanical sensors attached to the chin, lips, cheeks, and eyebrows, and electro-magnetic sensors on the supporting helmet structure
1992	Brad deGraf—Alive [43]	A real-time animation system with a special hand device with five plungers actuated by the puppeteer's fingers
1993	Acclaim	A realistic and complex two-character animation done entirely with motion capture

Optical system has complex hardware which requires 4 to 12 cameras to capture simultaneously. It tracks the reflective markers (sticking to the joints) in video streams to acquire 2D positions. Finally computer reconstructs the 3D motion information using computer vision principles. It can recuperate the drawbacks of the above two systems by high sampling rates and unrestricted movement, with the cost of high price, operating experience, unreliable tracking from self-occlusion, which are hard to be overcome.

The drawbacks in the above systems show that a motion capture system of low-cost, unrestrictive movement from scene and devices is essential, which is also the research issue in the intelligent animation.

1.3 Motion Editing and Reuse Techniques

As the motion capture systems are widely used, the requirement of editing and reusing motion (capture) data comes along. First, due to noise and restrictions from capture site, the captured motion is not accurate. Thus, motion data processing is necessary. Second, captured motion is not always suitable to the computer generated virtual environments. The data need to be adjusted to satisfy the requirements from applications. Finally, it is sound to reuse the motion capture data since the motion capture equipments are expensive in purchase and use. It also requires editing the data. It is important in intelligent computer animation research to edit and improve the reusability of motion data. One most commonly used technique is constraint-based motion editing method.

The most important characteristic in constraint-based motion editing is to treat some features of motion data and use requirements as constraints and preserve them in processing. These constraints can be divided into two classes: spatio-temporal constraints and temporal domain constraints. The former depicts a specific pose of character in some moment; the latter indicates that the editing results need to be natural and fluent.

Due to the hierarchical representation of motion capture data, each motion parameter describes motion through nonlinear relation constraints, which make motion editing highly difficult. Therefore, feasible constraint-based motion editing techniques all introduce inverse kinematics. Inverse kinematics has only two solution methods, analytical solution and numerical iterative solution. The advantage of analytical solution is to assure solution efficiency. But it is hard to obtain results when lacking constraints and not general for various concrete algorithms. Numerical iterative algorithm has good generality but needs expensive computation. It can be classified as non-constraint iterative solution or constraint iterative solution. The former is to optimize the object function in inverse kinematics by non-constraint methods. Such methods usually compromise between accurate positions of joint ends and other objectives. The latter is to use the accurate positions and rotational limitations of joints as hard constraints. But this method needs specific complex numerical optimization solution.

Current constraint-based motion editing methods can be divided into six categories with respect to the difference in temporal domain constraint processing. They will be described as follows.

1.3.1 Key-frame Editing

This kind of motion editing techniques force constraints on key-frames of motion data and solve it independently to obtain satisfactory key-frame poses, which can be used to produce other frames by interpolation. The effect of such techniques depends on the choice of key-frames. When the choice of key-frames is proper and adequate, natural editing results will be achieved. However, because the optimization is performed on each key-frame independently, such techniques themselves cannot guarantee results to be natural and fluent. Especially in processing motion capture data and motion data from simulating algorithms, results will be more severe. A representative of such methods was proposed by Bindingavale [44], which automatically chooses the extremes of motion data as key-frames, uses inverse kinematics to solve the key poses and finally obtains other poses by interpolation.

1.3.2 Motion Warping

One disadvantage of key-frame editing is that key-frames can be set in places that are not easy to edit. Therefore, researchers proposed a method called motion warping [45] or displacement mapping [46] to solve this problem. Different from key-frame editing techniques which interpolate among key-frames, this technique interpolates among the difference between target motion and original motion. The workflow of a classical motion warping system can be divided into two steps [47]: first using inverse kinematics to perform key-frame constraint optimization, then interpolating the difference to obtain the variation of other frames.

The advantage of such editing techniques is that key-frame can be positioned in any editable place. But there are still disadvantages: (1) lacking pose control ability for non key-frames; (2) decreasing control for temporal domain constraints, when increasing the number of key-frames for increasing the control over spatial domain.

1.3.3 Per-frame Editing

Per-frame editing performs on data sampled from motion data at high sampling rate. It is a special key-frame editing technique in a sense. However, these two techniques have major differences. Key-frame editing has a premise, i. e. the key-frames are defined on the essential time points, and therefore, as long as key-frames satisfy the requirements, the entire motion will also meet them. Regarding the per-frame motion editing, edited frames are samples at high sampling rate and have strong temporal correlations, with short time intervals. Although per-frame editing substantially is optimization by constraints from inverse kinematics, previous consecutive frames are considered in solution process in order to preserve the continuity in temporal domain. One important application for per-frame motion editing is real-time motion editing, e. g. live performance of computer puppets, where performer must drive puppet by computers in real-time. Other applications are computer games and virtual reality, in which characters must generate current pose with respect to current environment, without changing previous poses and knowing contents in upcoming frames. One representative of such techniques is motion retargeting system [48]. In each frame, constraints from inverse kinematics are linearly processed. In condition of ill-constraints, the system adjusts the original data to meet the constraints applied on the position of end joints. In this way, the generated new motion can proceed according to the path set by the performers and guarantee the fluency in temporal domain. Shin, et al. [49] proposed a more general real-time motion editing technique, in which Kalman filter was used to reduce noise and increase temporal fluency, and a high performance inverse kinematic operator was employed to satisfy the constraints of end joints. An important metric was also designed for switching reasonably between tracking the end joint positions and its angles.

1.3.4 Per-frame Motion Editing Combing Filters

After deciding how to edit each frame, the next step is the global processing for the editing results to solve or alleviate the problems such as burr, unevenness, etc. Until now, the second step work is accomplished by low pass filters. The first technique published is hierarchical motion editing technique [50], in which B-spline fitting is used to implement a low pass filter.

There is a dominant characteristic in filter aided per-frame motion editing techniques, i. e. two processes can be distinguished clearly: first using theory of inverse kinematics and adjusting every motion parameter to satisfy spatio-temporal constraints, then using signal processing techniques to satisfy spatio-temporal constraints. In the solution phase of spatial constraints, all motion parameters on a time point will be processed. In the solution phase of temporal constraints, a motion parameter spanning the entire temporal interval will be handled. The two phases do not take impact from each other into account, and therefore will counteract each other's effect. Consequently, the two processes are usually performed alternatively.

In theory, processing each motion parameter independently is not accurate. In the level of locality, grouped motion parameters are combined to represent rotation parameter. For example, it is meaningless for the angles represented by the parameters to perform filtering each parameter in Euler angles independently. But in practice, this method is feasible only if used for processing displacement. Recently, Lee and Shin [51] processed the angle data using FIR filters under exponential coordinate systems. However, this method cannot be applied in most of the filter aided per-frame motion editing techniques.

1.3.5 Spatio-temporal Constraint Based Motion Editing

Spatio-temporal constraint based motion editing technique is different from other constraint-based methods described above as it does not process each frame independently. spatio-temporal constraint here means processing a segment of motion at one time in computation. Different from the inverse kinematic operator in previous motion editing techniques, which processes independently each frame, the spatio-temporal constraint solves a sub-segment in the entire motion.

Spatio-temporal constraint method was first used to specify position for a specific character at a specific time and then to solve the optimum motion satisfying these constraints. In earlier works [52,53], physical laws were first applied to motion as constraints and energy consumed in muscle contraction for character motion alternation as objective function. Finally minimizing the energy function will obtain new motion. This synthesized motion can drive simple animation conforming to physical laws of single character.

Such kind of spatio-temporal motion editing techniques have both advantages and disadvantages. Although such techniques provide opportunities to describe motion features by defining constraints or objective functions, these descriptions are not always feasible. It is challenging to depict visual motion mathematically. However, so far, abstract features such as anger, elegance, etc. , are hard to describe. Furthermore, this kind of methods generates an entire new motion by solving a mathematical problem, which is usually a huge constraint optimization problem with highly complicated computation and low efficiency.

Gleicher [54-57] proposed a series of improvements of the above spatio-temporal constraint method. One of them is spatio-temporal constraint motion transformation [54]. Theoretically, the difference between this algorithm and traditional methods is that the former defines constraints closest to the original motion, while the latter defines by minimizing energy cost, which avoids the difficulty of depicting motion details. It is not necessary to describe a motion by constraints, instead by just providing a motion example. Till now, constraint optimization method has been applied to many motion editing tasks. Gleicher's spatio-temporal constraint based motion editing method [55] provides support to real-time interactive motion editing abilities. Another extremely valuable work is motion retargeting [56], which can automatically adjust the motion of one character to drive the motion of other characters. Besides, Gleicher also implemented motion path editing techniques [57], which could adjust motion by paths.

It is not an easy task to preserve motion attributes in a specific motion, such as realism, elegance, romance like singing in rain, etc. These high level motion attributes need to be preserved but hard to be achieved. In practice, defining high level motion attributes by mathematics is often restricted by capability and devotion. Even if motion attributes are encoded by constraint optimization methods, the solution is also highly difficult since complex optimization problems also accompany complex solutions.

1.3.6 Physical Property Based Motion Editing

Physical attributes provide some specific, useful constraints. Although some physical attributes can be defined as spatial constraints conveniently, some others are often neglected with regard to algorithm performance. The most obvious example is that Newton law is usually neglected, which is also one important reduction in spatio-temporal motion editing proposed by Gleicher. Dynamic constraints and energy law are hard for computation under the framework of spatio-temporal constraints.

However, dynamic constraints can be extremely important for some motions. Under such conditions, simplifying the problem by neglecting Newton law is not feasible. Therefore, Popovic and Witkin[58] proposed a physical attribute based motion editing technique, which simplified the problem by reducing the topological structure of characters and was different from Gleicher's method which neglected physical properties to enable a solution in constraint-based optimization. So far, there are few people investigating in-depth these physics based motion editing techniques. These techniques are never used in application and only used in experiments to expand single cycle walk to a full segment of motion.

1.4 Data-driven Animation Techniques

As the motion capture equipments become popular, large commercial body motion and facial expression databases have emerged and promoted the rapid development of data driven animation techniques. It becomes a focus of researchers to produce character animation easily, efficiently and even intelligently using existing human motion or expression data. We define intelligent character animation as a group of methods to produce character animation automatically or by simple user interaction of animators using existing body and expression data. It has characteristics of simplicity, efficiency and intelligence.

In recent years, many methods have been emerging in academic conferences and journals, which can be regarded as intelligent character animation. They can be divided into two classes according to their target problems: (1) data synthesis oriented methods, mainly about using existing data to satisfy new requirements of animators; (2) environment sensitive character animation methods, producing animation sequence accounting for objects in virtual scenes using existing data.

1.4.1 Data Synthesis Oriented Character Animation

The object of data synthesis oriented character animation is to synthesize new data satisfying requirements of animators through user interaction or automatically based on existing data. The input of such methods is one or more motion (expression) data, and the output is a segment of new motion (expression) data. There are many similar methods recently in various academic conferences and journals. Rather than citing all of them, we will introduce some of classical methods here.

In 2001, Gleicher [57] proposed a path editing algorithm, which represented the path of original motion as splines and further enabled animators to edit the splines for path editing. After obtaining new splines (path), which will be re-sampled, character local coordinate system will be remapped at each frame in order to map the original data to new path for satisfying requirements of animators.

In 2002, Kovar, et al. [59] introduced a concept of motion graph, which contained the connectivity of original data (frames) for generating new data by searching on the motion graph and guaranteeing continuous motion path.

In 2003, Kovar, et al. [60] illustrated a data structure named registration curves, which contained the time, coordinates and constraints information of original data. This data structure can be used to merge two input motion data by a group of interpolation algorithms to produce new motion data.

In 2004, Kovar and Gleicher [61] proposed a parameterized human motion generation method. Different from traditional fusion methods, which interpolate with known data and fusion parameters, Gleicher's method synthesizes new motion by constructing a parameter space to solve interpolation parameters of object motion from existing data and user assigned motion objective parameters.

In 2000, Brand and Hertzmann [62] tried to learn motion styles from motion data, added a multi-dimensional style variable to make style HMM as "style machine" and produced new motion data with specified styles. Likewise, Hsu, et al. [63] used a linear constant model to learn and model the difference in styles of data, which was used to change the style of new input data. Some research work about motion style synthesis can be found in [63—65]. In this book, the authors also propose a PCA subspace based style generation and editing method, in Chapter 7.

In 2004, an example based motion cloning method was proposed by Park and Shin [66]. The basic idea is to extract several key-frames in the original motion and let animators retarget these key-frames to new character model. Then based on the key-frame constraints, motion data cloning is achieved by dynamic time warping and motion retargeting methods to endow the remaining motion to the target model.

In 2002, Liu and Popovic [67] put forward a physical model based realistic human motion generation method. According to the input of several non-realistic key-frames by animators, realistic 3D character animation sequence was generated by combining dynamic model. Afterwards, Liu, et al.[68,69] published a series of papers about physical model based motion data generation, nonlinear optimization and physical model based style learning, physical model based multi-character motion generation, etc.

If readers intend to understand in-depth recent developments or one specific method, they can refer to conference proceedings of ACM SIGGRAPH, ACM Symposium on Computer Animation, Computer Animation and Social Agents (CASA), Computer Graphics International (CGI), Pacific Graphics, and international journals, such as ACM Transactions on Computer Graphics, Computer Animation and Virtual Worlds, etc.

1.4.2 Environment Sensitive Character Animation

The objective of environment sensitive character animation is to take existing motion (expression) data as input, combine non-life objects in virtual scenes, use motion editing and synthesis methods described above, to produce final animation sequence. It can be regarded as animation sequence synthesis from existing motion (expression) data and virtual scenes. These techniques are recently popular in computer animation field. Some typical works are introduced as follows.

Group Animation Generation

Group animation generation has been widely applied in commercial animation industry, e. g. battle scene in the movie "Troy". However, group animation techniques in commercial animation industry largely depend on the intervention of animators and are inefficient. Recently emerging group animation generation techniques will be briefly introduced here.

In 2005, Sung, et al.[70] proposed a target oriented fast group animation generation technique. Given the poses, positions and directions at specific time points as constraints, the system can rapidly and automatically produce the behavior animation of multiple characters and avoid the collision among characters and scene.

Treuille, et al.[71] suggested a real-time group animation model to generate animation in urban environment based on a continuous dynamic system. In this model, virtual characters are regarded as movable barriers and a continuous dynamic model and a global navigating algorithm are used to generate final animation. Likewise, in 2005 Shao[72] depicted in their paper a group animation generation method for virtual scene. Different from the dynamic model by Treuille, et al., Shao used artificial life to produce group animation and comprehensive model for modeling each virtual character to enable perception, action and cognition. Other work about crowd animation can be found in [73—78].

Motion Planning in Animation

The objective of motion planning in animation is to solve the character animation in complex scene or with certain tasks. Two classes of problems are well studied: (1) character animation in complex scene; (2) task oriented character animation.

Given constraints from animators, such as motion start and motion end, motion planning is to generate character animation automatically in a complex scene based on the motion editing and reuse techniques.

In 2003, Choi, et al. [78] employed probabilistic roadmaps to generate character animation in complex scene based on existing data. First, a probabilistic model is used to model the footsteps of virtual character in scene and used as a basis to retrieve satisfactory motion data from motion capture database. Final animation sequence is generated after operations such as editing, synthesis, etc.

Lau and Kuffner [79] proposed a method to generate realistic human motion animation by behavior planning in virtual scenes. First, existing motion data segments are abstracted to high level behaviors and associated with behavior finite-state machine, which defines the motion capability of virtual characters. In runtime, a dynamical programming algorithm based FSM global search can produce behavior sequence for character to move to a specific position. This method can produce single or multiple characters animation in complex and dynamic virtual scenes.

Similarly, the authors of this book proposed a character animation method based on motion planning and motion scripts, which will be described in Chapter 8.

Virtual task oriented character animation is to automatically synthesize a sequence of animation to accomplish and use assigned task using motion editing, reuse and synthesis techniques based on existing data. Yamane et al. [80] proposed an object operation task oriented animation production method and gave an example of moving objects. First, path planning is used to compute the motion path satisfying geometric, kinematical, pose constraints at each point of the path. Then final motion sequence will be obtained by retrieving similar motion in database and solving inverse kinematic equations.

1.5 Intelligent Animation

1.5.1 Characteristics and Requirements of Intelligent Animation

In order to overcome the drawbacks in existing solutions and to get high production rate, high intelligence and high reality, a new animation technique — intelligent animation is developing rapidly.

The production of animation is a complicated flow, including acquisition, reuse of animation materials and animation pipeline. Compared with the traditional animation, intelligent animation has its own characteristics.

Human and lives, whose behaviors are indispensable elements, are the soul of animation. Therefore, intelligent animation requires animation elements to be acquired widely, conveniently, accurately and rapidly in body behavior and facial expression data.

The available key-frame and motion capture techniques show great deficiency in data reuse. Especially in the key-frame animation, with respect to specific animation scene and behaviors, animators draw key-frames by hand, which could hardly be used in other animation productions. In the aspect of motion and expression capture, commercial software to some extent support the data reuse which relies on the animators' handcraft. However, one of the characteristics in intelligent animation production technique is to reuse the animation materials efficiently and intelligently in order to improve the animation production rate.

In the workflow of animation production, traditional animation techniques require a great deal of manual intervention. Animators need to make a great effort to produce final animation, such as setup and adjustment of key-frames, character motion path planning in specific scenes and manual stitching of motion data, which reduce the animation production efficiency largely. Intelligent animation introduces artificial intelligence to traditional animation techniques to ease animators from heavy manual work and realize the intelligent production of animation works.

1.5.2 Overview of Video-based Intelligent Animation Techniques

In the field of computer animation, human behavior with facial expressions is an essential direction. Traditional character animation is also called as articulation animation. The movement of articulations is controlled by forward or inverse kinematics. But as the number of joints grows, the time complexity of finding the solution becomes extremely high. The articulation animation requires not only tedious labor of animators, but also high computation cost. The final character movement suffers from lack of reality because it is only simulation of real motion. Motion capture based animation becomes increasingly popular. When actors/actresses perform a specific action, the sensors attached to the body joints are returned to computer. Realistic body motion can be obtained by applying the data on 3D human models. Although this technique can generate more realistic animation than the articulation animation, the acquisition of motion information is expensive and requires special workplace or cables attached on the human body, which restricts the freedom of the performer's motion, leading to some motion distortion.

The above techniques are all from a model perspective to generate animation and are self-contained. A complete animation should traverse various phases from physical modeling, coloring, scene setup, motion setup to animation generation. For an experienced animator, he can choose a most similar one from his past products and revise it to get a desired result. This is not always a good solution especially for those who just step into the field of animation. Current animation systems are built on the base of mathematical models, in which the tuning of parameters is tedious and obscure. It is hard to master such a system even for experienced computer graphics people, to say nothing of an artist who never uses a computer. The obscure terms and operations greatly restricted the animation systems from wide application. Based on the analysis, animators have to think differently in two mindsets, one from an artist perspective, and the other from computer animation, the latter of which largely reduces the creativity of animators and decreases the production efficiency.

From a view of artificial intelligence, people learn new things first by imitating. It is also true in animation making. Examples play an important role in animation making. An animator often employs his previous successful products to realize his new idea. The revision of the previous products leads to satisfaction of new requirements. The situation is different when there are no available examples. One has to start from the very beginning. For one animator, the production quantity and scale are all limited. Furthermore, there are few published animation productions (including producing process). Therefore, there are few examples for reference.

The researchers in computer animation field neglected an extremely rich source, i. e. video. Human beings have recorded abundant videos ever since the birth of movie. It is an under-determined problem to exploit these videos for the purpose of computer animation. How to provide the interface between video and computer animation? The interface is definitely not just to read video content to computers, which are solved by many mature commercial products. The interface here is to enable computers to acquire analyzed high-level structural information from video, such as motion trajectories, deformation methods, camera positions, foreground, background, scene setup, style, etc.

We can see that the cost of acquiring motion from video is extremely low (almost zero) by just using a movie script and corresponding software to make it possible to obtain 3D motion. In the near future, we can imagine such exciting applications:

- In a movie, an infant is walking in a Chaplinesque way, which is not imitating but derived from the master's real movement, which could only be appreciated from several old films in the past;
- An ordinary family one day makes an animation film, where all the motion is from the family members, which was far from family enter-

tainment due to the expensive hardware requirement in the past.

Therefore, the basic idea of video-based intelligent animation techniques is: given a segment of video, extracting 3D motion and expression information based on human body movement in the video stream, building a motion library for data reuse and developing a set of intelligent tools for character animation. In this way animators can generate new character animation based on the existing motion information. The video-based intelligent animation techniques consist of several aspects: (1) video-based body motion and facial expression data acquisition; (2) 3D body motion and expression database construction and reuse; (3) intelligent generation of character animation.

It is a challenging task to acquire realistic body motion and expression from video stream. The contribution of this research path is to propose video as a new character animation material and a framework to implement these techniques. Whether the comedy films of Chaplin or scudding shot of Karl Lewis can be used as animation material means animators can use more than ever motion information. Video-based character animation and expression are very convenient and intuitive. Users only need to make some annotations on the first frame, and then human motion can be viewed from any viewpoints. On the other hand, from a perspective of motion acquisition, this method removes the restrictions in the previous methods, and finally establishes the 3D motion model in the perspective view, which can be used in various 3D scene demanding applications such as character animation, virtual reality, etc.

References

1. Guenter B, Parent R. Computing the arc length of parametric curves. IEEE Computer Graphics and Applications, 10(3): 72-78, 1990.
2. Watt A, Watt M. Advanced animation and rendering techniques. Addison-Wesley Publishing Company, 1992.
3. Steketee SN, Badler NI. Parametric keyframe interpolation incorporating kinetic adjustment and phrasing control. Computer Graphics, 19 (3): 255-262, 1985.
4. Kochanek DHU, Bartels RH. Interpolating splines with local tension, continuity and bias control. Computer Graphics, 18(3): 245-254, 1984.
5. Brotman LS, Netravali AN. Motion interpolation by optimal control. Computer Graphics, 22(4): 179-188, 1988.
6. Shoemake K. Animating rotation with quaternion curves. Computer Graphics, 19(3): 245-254, 1985.
7. Duff T. Splines in animation and modeling. In: SIGGRAPH'86,

ACM Press, 1986.
8. Pletincks D. Quaternion calculus as a basic tool in computer graphics. The Visual Computer, 5: 2-13, 1989.
9. Barr AH, Currin B, Gabriel S, Hughes JF. Smooth interpolation of orientations with angular velocity constraints using quaternions. Computer Graphics, 26(2): 313-320, 1992.
10. Maiocchi R, Pernici B. Directing and animated scene with autonomous actors. The Visual Computer, 6: 359-371, 1990.
11. Korein JU, Badler NI. Techniques for generating the goal-directed motion of articulated structures. IEEE Computer Graphics and Applications, 2(3): 71-81, 1982.
12. Girard M, Maciejewski AA. Computational modeling for the computer animation of legged figures. Computer Graphics, 19(3): 263-270, 1985.
13. Girard M. Interactive design of 3D computer-animated legged animal motion. IEEE Computer Graphics and Applications, 7(6): 39-51, 1987.
14. Isaacs PM, Cohen MF. Controlling dynamics simulation with kinematic constraints, behaviour functions and inverse dynamics. Computer Graphics, 21(4): 215-224, 1987.
15. Boulic R, Thalmann NM, Thalmann D. A global human walking model with real-time kinematic personification. The Visual Computer, 6(6): 344-358, 1992.
16. Philips CB, Badler NI. Interactive behaviour for bipedal articulated figures. Computer Graphics, 25(4): 359-362, 1991.
17. Wilhelms J, Barsky BA. Using dynamics analysis for the animation of articulated bodies such as human and robots. In: Graphics Interface'85, pp. 97-104, Springer-Verlag, 1985.
18. Wilhelms J. Using dynamics analysis for animation of articulated bodies. IEEE Computer Graphics and Applications, 7(6): 12-27, 1987.
19. Wilhelms J. Towards automatic motion control. IEEE Computer Graphics and Applications, 7(4): 11-22, 1987.
20. Armstrong WW. Recursive solution to the equations of motion of an n-link manipulator. In: Proceedings of the Fifth World Congress on the Theory of Machines and Mechanisms. pp. 1343-1346, 1979.
21. Armstrong WW, Green MW. The dynamics of articulated rigid bodies for purposes of animation. The Visual Computer, 1: 231-240, 1985.
22. Witkin A, Fleischer K, Barr AH. Energy constraints on parameterized models. Computer Graphics, 21(4): 225-232, 1987.
23. Moore M, Wilhelms J. Collision detection and response for computer animation. Computer Graphics, 22(4): 289-298, 1988.

24. Zeltzer D. Motor control techniques for figure animation. IEEE Computer Graphics and Applications, 2(9): 53-59, 1982.
25. Bruderlin A, Calvert TW. Goal-directed, dynamics animation of human walking. Computer Graphics, 23(3): 233-242, 1989.
26. Bruderlin A, Teo CG, Calvert TW. Procedural movement for articulated figure animation. In: Proc of CAD/Graphics'93, pp. 141-146, International Academic Publishers, 1993.
27. Boulic R, Thalmann D. Combined direct and inverse kinematic control for articulated figure motion editing. Computer Graphics Forum, 11(4): 189-202, 1992.
28. Raibert MH, Hodgins JK. Animation of dynamic legged locomotion. Computer Graphics, 25(4): 249-358, 1991.
29. McKenna M, Zelter D. Dynamic simulation of autonomous legged locomotion. Computer Graphics, 24(4): 29-38, 1990.
30. Witkin A, Kass M. Spacetime constraints. Computer Graphics, 22 (4): 159-168, 1988.
31. Cohen MF. Interactive spacetime control for animation. Computer Graphics, 26(2): 293-302, 1992.
32. Liu Z, Gortler SJ, Cohen MF. Hierarchical spacetime control. Computer Graphics, 29(4): 35-42, 1994.
33. Ngo JT, Marks J. Spacetime constrained revisited. Computer Graphics, 27(4): 343-350, 1993.
34. Sims K. Evolving virtual creatures. Computer Graphics, 29(4): 15-22, 1994.
35. Sturman DJ. A brief history of motion capture for computer character animation. Character motion system. In: SIGGRAPH ' 94, Course 9, ACM Press, 1994.
36. Calvert TW, Chapman J, Patla A. Aspects of the kinematic simulation of human movement. IEEE Computer Graphics and Applications, 2(9): 41-50, 1982.
37. Ginsberg CM, Maxwell D. Graphical marionette. In: Proc. ACM SIGGRAPH/SIGART Workshop on Motion, pp. 172-179, ACM Press, 1983.
38. Robertson B. Mike, the talking head. Computer Graphics World, 11 (7): 15-17, 1988.
39. Walters G. The story of Waldo C. Graphic. In: SIGGRAPH'89, pp. 65-79, ACM Press, 1989.
40. Walters G. Performance animation at PDI. In: SIGGRAPH'93, pp. 40-53, ACM Press, 1993.
41. Kleiser J. Character motion systems. In: SIGGRAPH'93, pp. 33-36, ACM Press, 1993.
42. Tardif H. Character animation in real time. In: SIGGRAPH Panel Proceedings, I: Reports from the Field, ACM Press, 1991.

43. Robertson B. Moving pictures. Computer Graphics World, 15(10): 38-44, 1992.
44. Bindiganavale RN. Building parameterized action representations from observation. Ph. D. thesis, University of Pennsylvania, 2000. [Appears as Technical Report MS-CIS-00-17].
45. Witkin A, Popovic Z. Motion warping. In: SIGGRAPH'95, pp. 105-108, ACM Press, 1995.
46. Bruderlin A, Williams L. Motion signal processing. In: SIGGRAPH'95, pp. 97-104, ACM Press, 1995.
47. Choi KJ, Park SH, Ko HS. Processing motion capture data to achieve positional accuracy. Graphical Models Image Process, 61(5): 260-273, 1999.
48. Choi KJ, Ko HS. On-line motion retargeting. Journal of Visualization and Computer Animation, 11: 223-243, 2000.
49. Shin HJ, Lee J, Gleicher M, Shin SY. Computer puppetry: an importance-based approach. ACM Transactions on Graphics, 20(2): 67-94, 2001.
50. Lee J, Shin SY. A hierarchical approach to interactive motion editing for human-like figures. In: SIGGRAPH'99, pp. 39-48, ACM Press, 1999.
51. Lee J, Shin SY. Multiresolution motion analysis and synthesis. Technical Report CS-TR-2000-149, Computer Science Department, KAIST, 2000.
52. Witkin A, Kass M. Spacetime constraints. Computer Graphics (SIGGRAPH'88), 22(1): 159-168, 1988.
53. Cohen MF. Interactive spacetime control for animation. Computer Graphics (SIGGRAPH'92), 26(2): 293-302, 1992.
54. Gleicher M, Litwinowicz P. Constraint-based motion adaptation. Journal of Visualization and Computer Animation, 9: 65-94, 1998.
55. Gleicher M. Motion editing with spacetime constraints. In: 1997 Symposium on Interactive 3D Graphics, pp. 139-148, ACM Press, 1997.
56. Gleicher M. Retargeting motion to new characters. In: SIGGRAPH'98, pp. 33-42, Addison Wesley, 1998.
57. Gleicher M. Motion path editing. In: SI3D 2001, pp. 195-202, ACM Press, 2001.
58. Popovic Z, Witkin A. Physically based motion transformation. In: SIGGRAPH'99, pp. 11-20, ACM Press, 1999.
59. Kovar L, Gleicher M, Pighin F. Motion graphs. In: SIGGRAPH'02, pp. 473-482, ACM Press, 2002.
60. Kovar L, Gleicher M. Flexible automatic motion blending with registration curves. In: SIGGRAPH'03, pp. 214-224, ACM Press, 2003.

61. Kovar L, Gleicher M. Automated extraction and parameterization of motions in large data sets. In: SIGGRAPH'04, pp. 559-568, ACM Press, 2004.
62. Brand ME, Hertzmann A. Style machines. In: SIGGRAPH 2000, pp. 183-192, ACM Press, 2000.
63. Hsu E, Pulli K, Popovic J. Style translation for human motion. ACM Transactions on Graphics, 24(3): 1082-1089, 2005.
64. Grochow K, Martin SL, Hertzmann A, Popovic Z. Style-based inverse kinematics. In: SIGGRAPH'04, pp. 522-531, ACM Press, 2004.
65. Torresani L, Hackney P, Bregler C. Learning to synthesize motion styles. Snowbird Learning Workshop, 2006.
66. Park MJ, Shin SY. Example based motion cloning. Computer Animation and Virtual Worlds, 15: 245-257, 2004.
67. Liu CK, Popovic Z. Synthesis of complex dynamic character motion from simple animation. In: SIGGRAPH'02, pp. 408-416, ACM Press, 2002.
68. Liu CK, Hertzmann A, Popovic Z. Learning physics-based motion style with nonlinear inverse optimization. ACM Transactions on Graphics, 24(3): 1071-1081, 2005.
69. Abe Y, Liu CK, Popovic Z. Momentum-based parameterization of dynamic character motion. Graphical Models, 68 (2): 194-211, 2006.
70. Sung M, Kovar L, Gleicher M. Fast and accurate goal-directed motion synthesis for crowds. In: 2005 SIGGRAPH/EuroGraphics Symposium on Computer Animation, pp. 291-300, ACM Press, 2005.
71. Treuille A, Cooper S, Popovic Z. Continuum Crowds. In: SIGGRAPH'06, pp. 1160-1168, ACM Press, 2006.
72. Shao W. Animating autonomous pedestrians. Ph. D. thesis, New York University, 2006.
73. Liu F, Zhuang Y, Luo Z, Pan Y. Group animation based on multiple autonomous agents. Chinese Journal of Computer Research and Development, 41(1): 104-110, 2004.
74. Musse SR, Thalmann D. Hierarchical model for real time simulation of virtual human crowds. IEEE Transactions on Visualization and Computer Graphics, 7(2): 152-164, 2001.
75. Massive Software, Inc. 3D animation system for crowd-related visual effects. http://www.massivesoftware.com, 2005.
76. Lamarche F, Donikian S. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. Computer Graphics Forum, 23(3): 509-518, 2004.
77. Sung M, Chenney S, Gleicher M. Scalable behaviors for crowd simulation. Computer Graphics Forum (EuroGraphics 2004), 23(3):

519-528, 2004.

78. Choi MG, Lee L, Shin SY. Planning biped locomotion using motion capture data and probabilistic roadmaps. ACM Transactions on Graphics, 22(2): 182-203, 2003.

79. Lau M, Kuffner J. Behavior planning for character animation. In: SIGGRAPH/ EuroGraphics Symposium on Computer Animation, ACM Press, 2005.

80. Yamane K, Kuffner JJ, Hodgins JK. Synthesizing animations of human manipulation tasks. ACM Transactions on Graphics (SIGGRAPH'04), 23(3): 532-539, 2004.

2

Natural Video-based Human Motion Capture

The ability to extract human motion data from video is very important in intelligent animation. Videos are abundant in our daily life and the cost to make videos is becoming lower. If the 2D/3D human motion data required for producing animations can be easily and rapidly extracted from existing videos, the cost will be reduced and the efficiency will be improved significantly.

This chapter will discuss in detail the key techniques in human motion capture based on natural video. By natural video we mean monocular video taken by ordinary cameras without special limitations on the sets and target objects (i. e. movies, dramas, homemade DV clips). The purpose of naming "natural video" is to distinguish it from lab video, in which markers are attached deliberately. Natural videos have the following properties:

- They are the monocular and the cameras are not calibrated in advance;
- They don't have constraints on the capturing environments and the backgrounds are typically complex;
- No special markers are attached to foreground objects and frequent self-occlusion exists during movements;

Therefore, capturing 3D human motion from this kind of videos is challenging, and the existing techniques are often intricate and complex in theory. In addition to the discussion in the theoretical level, this chapter will also offer the demonstration of VBHAS V1. 0, the video-based human motion animation system developed by the authors.

Technically speaking, two main tactics exist in recovering 3D human motion sequences from natural videos: (1) 3D human motion recovery by feature tracking, which tracks feature points in video sequence then reconstructs 3D coordinates for the tracked features; (2) 3D human motion recovery by silhouettes, which first extracts silhouette and identifies head and limbs position then searches for a corresponding pose under the guidance of a 3D human model. Below we will elaborate on the two techniques.

2.1 Human Motion Capture Based on Feature Tracking

Identifying and tracking objects in image sequence is a classic topic in computer vision. Two approaches to this topic have been proposed, which usually following three steps: (1) feature extraction in video frames, such as body part segmentation, joint detection and identification; (2) correspondence between the features of every frame; (3) recovery of 3D human structure and motion from feature correspondences. O'Rourke and Badler [1] conducted 3D human motion analysis by mapping the input images to a volumetric model.

The systems of Hogg [2] and Rohr [3] were specialized to a one-degree-of-freedom walking model, in which edge and line features are extracted from images and matched to a cylindrical 3D body model. Chen and Lee [4] used 17 line segments and 14 joints to represent the human skeleton model, and various constraints were imposed on the basic analysis of the gait. Bharatkumar, et al. [5] also used stick figures to model the lower limbs of the human body, and their goal was to construct a general model for gait analysis in human walking. Akita [6] focused on the model-based motion analysis for real image sequence, and they used a key-frame sequence of stick figures to indicate the approximate order of the motion and spatial relationships between the body parts. Bregler and Malik [7] recovered the 3D human motion under orthographic projection by marking the body segments in an initial frame. For the special complexity of human motion, the existing research methods laid considerable limitations, such as a uniform and quiescent background, parallelism of human motion direction to the image plane, and skintight clothing of human [8].

Considering human body as a complex articulated rigid model, it is very difficult to identify different body parts directly from a single image even with prior knowledge. Therefore, identifying human body in the first frame by manual labeling is a practical approach. If we don't adopt manual labeling for the first frame, then the system must solve the following two problems:

- The system must automatically identify the outer contour of human body in the image, which is a very difficult task with the current image segmentation techniques;
- The system must segment the area in the contour into different parts and assign semantics such as left thigh, right thigh, chest to the body parts, which is also a difficult task because how to derive from which part of human body as a specific area is not well known yet.

Manual labeling of the first frame requires users click at the positions of human body joints on the image, which is not very hard for the users. After labeling the first frame manually, movements of different body parts can be tracked in the following frames automatically.

Due to its relatively rare self-occlusion, *head* is a good choice for extracting color features. Besides, once the head rectangle is determined, *neck* is ready to be determined. Similarly, once an *upper arm* is determined, the *elbow* is also known and it is only required to identify the *hand* to determine the *forearm*. Therefore, it is practical to start from *head* and track the body in a downward order. We approximate the *head's* movement as movement in straight line with variable accelerations. First, we predict its movement by Kalman filter. Then we look for the correspondence of body parts between frames using image patch matching.

2.1.1 Human Skeleton Model

The 3D human body can be considered as a rigid collection of segments connected by joints. For example, an upper limb is formed by two rigid segments: upper arm and lower arm connected by elbow joint. If we use a line to represent such a rigid segment and simplify the human movement to the movement of human skeleton model, we can get a 3D human skeleton model as in Fig. 2.1. The model contains 16 joints, and in this chapter we call these joints as 3D feature points. The rigid segment represented by lines cannot have deformation during movements. The segment proportions can be obtained from anatomy knowledge. We don't consider the individual variation of these proportions.

The projection of 3D feature points on the 2D plane is called 2D feature points. The topological relations in the 3D skeleton model do not change during projection, but the length of segments will change. In the 2D tracking, we use rectangles to represent the projection of a segment, as in Fig. 2.2. For head, our experiments show that rectangle works better than circle. Besides, when the whole body is in the camera's view range, typi-

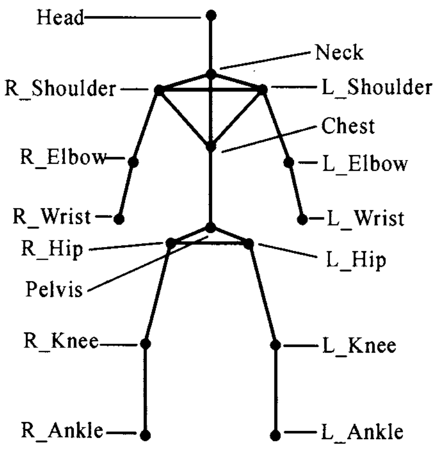


Fig. 2.1 Human skeleton model

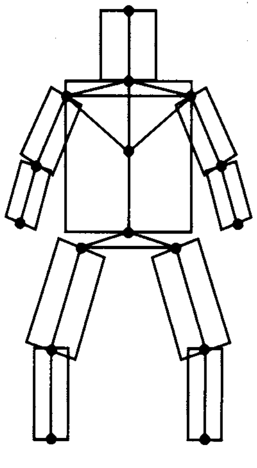


Fig. 2.2 Rectangles in 2D human body

cally, the areas occupied by individual body segments are small, so it is desirable to simplify these segments to rectangles. The lengths of the rectangles are the lengths of the projected segments and their widths can be determined from the prior knowledge. After user manually labels the first frame, the rectangles in this first frame are also known. Therefore, if we can find the new position of these rectangles in the following frames, we also get the 2D movement of the skeleton on the images.

2.1.2 Feature Tracking in 2D Image Sequence

Because there is little self-occlusion on a human head, its color information can be acquired easily. After the head block is tracked, one feature point of trunk, the neck, is also fixed. So, beginning with head, we track every body part from top to bottom. Now we detail the tracking of head, trunk and limb respectively.

For every frame in the sequence, the head may move toward any directions in the next frame. If a local search mode is used, the result may be not global optimal. If a global search mode is used, it suffers low efficiency. So we adopt the combination of these two. To reduce the search area of head point in the next frame, we introduce Kalman filter based global motion model to predict the motion of head point. Then in order to fix on the head accurately, we select a search path to do morph-block based match around the predicted point. Our experiments show that using Kalman filter to predict the head point has a good performance.

Regarding the sequence of motion images as a dynamic system, the head point can be described by the following equation:

$$\boldsymbol{P}=\boldsymbol{P}'+\boldsymbol{\eta} \tag{2-1}$$

where the coordinate $\boldsymbol{P}=(x, y)^{\mathrm{T}}$ is the tracked head point, $\boldsymbol{P}'$ is the actual coordinate, and $\boldsymbol{\eta}$ is a 2D Gaussian random noise with mean value $\mathbf{0}$ and covariance $\boldsymbol{R}$. We use 3rd polynomial to represent the motion trajectory of point $\boldsymbol{P}$. The state vector is defined as:

$$\boldsymbol{S}=(\boldsymbol{P},\boldsymbol{P}',\boldsymbol{P}'') \tag{2-2}$$

where $\boldsymbol{P}'=(x', y')^{\mathrm{T}}$ and x', y' represent the velocity of point $\boldsymbol{P}$ in the x, y directions respectively. $\boldsymbol{P}''=(x'', y'')^{\mathrm{T}}$ and x'', y'' represent the acceleration of point $\boldsymbol{P}$ in the x, y directions respectively. The state equation is defined as:

$$\boldsymbol{S}(k+1) = \boldsymbol{F}\cdot\boldsymbol{S}(k) + \boldsymbol{G}\cdot\boldsymbol{n}(k) \tag{2-3}$$

where

$$\boldsymbol{F}=\begin{bmatrix} \boldsymbol{I}_2 & \boldsymbol{I}_2\cdot T & \frac{1}{2}\boldsymbol{I}_2\cdot T^2 \\ \mathbf{0}_2 & \boldsymbol{I}_2 & \boldsymbol{I}_2\cdot T \\ \mathbf{0}_2 & \mathbf{0}_2 & \boldsymbol{I}_2 \end{bmatrix}, \quad \boldsymbol{G}=\begin{bmatrix} \frac{1}{2}\boldsymbol{I}_2\cdot T^2 \\ \boldsymbol{I}_2\cdot T \\ \boldsymbol{I}_2 \end{bmatrix},$$

$k=0,1,2,\cdots$ represents the serial number of the frame, $\boldsymbol{I}_2$ is a 2×2 unit matrix, $\boldsymbol{0}_2$ is a 2×2 zero matrix, and T is the time interval between frames. $\boldsymbol{n}(k)=(n_x(k), n_y(k))^{\mathrm{T}}$ describes the acceleration noise in the x, y directions. Suppose $\boldsymbol{n}(k)$ conform to the Gaussian distribution with even $\boldsymbol{0}$ and covariance $\boldsymbol{Q}$. This state equation shows that point $\boldsymbol{P}$ is doing varied-acceleration linear motion in all the x, y directions. In practice, we track the coordinate of point $\boldsymbol{P}$, i. e. $\boldsymbol{X}(k)=\boldsymbol{P}(k)$. So the measurement equation is:

$$\boldsymbol{X}(k) = \boldsymbol{H}\cdot\boldsymbol{S}(k) + \boldsymbol{\eta}(k) \tag{2-4}$$

where $\boldsymbol{H}=[\boldsymbol{I}_2, \boldsymbol{0}_2, \boldsymbol{0}_2]$ is a 2×6 matrix. In the above conditions, we get the recursive equations of Kalman filter as follows.

State vector prediction equation:

$$\boldsymbol{S}_k' = \boldsymbol{F}\cdot\boldsymbol{S}_k \tag{2-5}$$

State vector covariance prediction equation:

$$\boldsymbol{P}_k' = \boldsymbol{F}\cdot\boldsymbol{P}_{k-1}\cdot\boldsymbol{F}^{\mathrm{T}} + \boldsymbol{G}\cdot\boldsymbol{Q}\cdot\boldsymbol{G}^{\mathrm{T}} \tag{2-6}$$

Kalman filter gain matrix:

$$\boldsymbol{K}_k = \boldsymbol{P}_k'\cdot\boldsymbol{H}^{\mathrm{T}}\cdot(\boldsymbol{H}\cdot\boldsymbol{P}_k'\cdot\boldsymbol{H}^{\mathrm{T}} + \boldsymbol{R})^{-1} \tag{2-7}$$

State vector covariance update equation:

$$\boldsymbol{P}_k = \boldsymbol{P}_k' - \boldsymbol{K}_k\cdot(\boldsymbol{H}\cdot\boldsymbol{P}_k'\cdot\boldsymbol{H}^{\mathrm{T}} + \boldsymbol{R})\cdot\boldsymbol{K}_k \tag{2-8}$$

State vector update equation:

$$\boldsymbol{S}_k = \boldsymbol{S}_k' + \boldsymbol{K}_k\cdot(\boldsymbol{X}_k - \boldsymbol{H}\cdot\boldsymbol{S}_k') \tag{2-9}$$

Above we have applied the Kalman filter to predict the possible position of head point in the next frame. Then in order to fix on the head accurately, we choose a search path (see Fig. 2. 3) to do morph-block based match around the predicted point.

Because the head and the neck points have been located in the first frame, the height m of the head block is the distance between these two points and the proportion of height m to width n can be acquired in anatomy. The color information of $m\times n$ pixels in the block is saved as the color model for the matching of subsequent frames. Since the head block in the image is the projection of human head, the head motion will change the shape of projection. For example, the head block becomes larger, which is likely to happen when human is moving toward the camera. So, the block match must be processed between morph-blocks. Therefore, we propose a weighted morph-block similarity algorithm based on sub-pixel.

Define a feature morph-block $A=\{(x, y), m, n, \theta\}$ (see Fig. 2. 4), where (x,y) is the intersection of one side and the middle line, m is the height of block A, n is the width of block, and θ is the angle between the

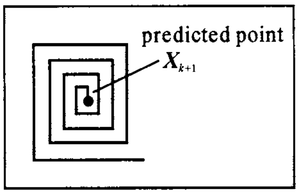


Fig. 2.3 Local search path

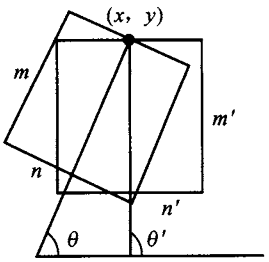


Fig. 2.4 Two feature morph-blocks

middle line and x axis. Now there are a reference block $A=\{(x,y),m,n,\theta\}$ and a comparative block $A'=\{(x',y'),m',n',\theta'\}$. To calculate their similarity, we use a algorithm as follows.

Step 1 If $m\times n < m'\times n'$, then *row*$=m$, *column*$=n$; else *row*$=m'$, *column*$=n'$.

Step 2 In block A we depict *column* and *row* pieces of gridding lines evenly in the direction of arctan θ and arctan $(-1/\theta)$ respectively. The intersection of any two gridding lines is named as sub-pixel X_{ij} $(0\leqslant i<m,\ 0\leqslant j<n)$. Then we use quadric linear interpolation to compute the color of every sub-pixel, X_{ij}[Red], X_{ij} [Green], X_{ij} [Blue].

Step 3 In block A' we depict *column* and *row* pieces of gridding lines evenly in the direction of arctan θ' and arctan $(-1/\theta')$ respectively. Then linear interpolation is used to compute the color of every sub-pixel, X'_{ij}[Red], X'_{ij}[Green], X'_{ij}[Blue].

Step 4 Calculate:

$$\begin{aligned} diff_{ij} = & W_R \cdot |X_{ij}[\text{Red}]-X_{ij}'[\text{Red}]| + \\ & W_G \cdot |X_{ij}[\text{Green}]-X_{ij}'[\text{Green}]| + \\ & W_B \cdot |X_{ij}[\text{Blue}]-X_{ij}'[\text{Blue}]| \end{aligned} \tag{2-10}$$

$$S = 1/(W_1 \cdot \sum_{(i,j)\in b_1} diff_{ij} + W_2 \cdot \sum_{(i,j)\in b_2} diff_{ij}) \tag{2-11}$$

where W_R, W_G, W_B represent the weights of each element in RGB, b_1, b_2 represent the two regions divided in the block, and W_1, W_2 represent the weight of each region in the whole block. In the case of the head, we define the center region as b_1 and the marginal region as b_2, respectively:

$$\begin{cases} (i,j)\in b_1, \text{ if } m/4\leqslant i\leqslant(3/4)m \text{ and } n/4\leqslant(3/4)n \\ (i,j)\in b_2, \text{ otherwise} \end{cases} \tag{2-12}$$

Here we have $W_1>W_2$. This weighted morph-block similarity measure is based on the observation that the marginal region of head has a more salient change of color in motion; while the center region has a relative smaller change. S is used to represent the similarity of two morph-blocks.

Note that the 3D human skeleton of frame $t-1$ has already been established when human joint is tracked in frame t. Now we introduce how to predict the initial height of head block in frame t using the 3D human motion. The model of projected image height of head is illustrated in Fig. 2.5. H is the actual height of head in the 3D human model, f is the focal length of the camera, and O is the optical center of the camera. A point (x_t, y_t) in frame t is related to point (X_t, Y_t, Z_t) in the 3D camera coordinate system by $(x_t, y_t) = (X_t \cdot f/Z_t, Y_t \cdot f/Z_t)$, where Z_t is the distance of the head from the camera in frame t. Thus we have:

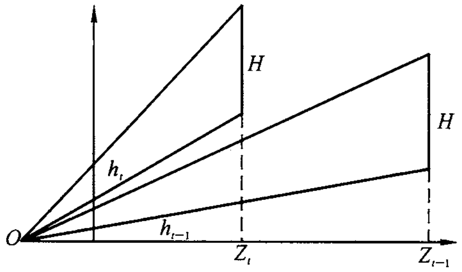


Fig. 2.5 Prediction of head height

$$h_t = H \cdot f/Z_t \tag{2-13}$$

where h_t is the height of head in frame t. Assume the head is approaching or moving away from the camera at a locally constant velocity v, i.e.

$$Z_t = Z_{t-1} + v \cdot T \tag{2-14}$$

Using equation (2-14) to substitute Z_t in equation (2-13), we get the initial height h in frame t.

For a frame sequence, the tracked head block in current frame is defined as a reference block A, and the head block in the next frame as a comparative block A'. θ' is set in the range of $[\theta-\Delta\theta, \theta+\Delta\theta]$. By the previous prediction algorithm we get an estimated height of the head h_t, and set m' in the range of $[h_t-\Delta m, h_t+\Delta m]$. Since the height and width of head zoom in proportion, n' is set in $[h_t(n/m)-(n/m)\Delta m, h_t(n/m)+(n/m)\Delta m]$. Starting from the predicted point, for every point (x, y) on the search path, we form a block A' by $\{(x,y), m', n', \theta'\}$ and calculate its similarity with the head block of current frame A. The system records the block A' which has the largest similarity. After finding the largest similarity, the search process will continue until it does not find a block which has a larger similarity on the search path of the next one circle. If it does, repeat the process mentioned in the last sentence. In the end, the last recorded A' is the head block in the next frame. And for the self adaptability of color model, linear weight is utilized to update the color model[9].

The tracking of *trunk* and *limb* also depends on the above algorithm. But we must pay attention to other two problems. Firstly, because of the large limb motion from frame to frame, we introduce a prediction mechanism to estimate the possible limb position in the next frame. As the example of thigh, the relative angle from knee to hip is preserved for every frame. While doing prediction, we calculate the average value of such angles in the previous two frames, use it as the initial angle θ', and fix on θ' in the scope of $(\theta'-\Delta\theta, \theta'+\Delta\theta)$. Our experiment shows that this prediction mechanism can reduce the search area of block match for the large motion. Secondly, we show how to deal with self-occlusion in the tracking of limb. For example, there is relative small similarity in the block match of an upper limb when in one frame the trunk occludes that upper limb. But the similarity will be larger as soon as the occlusion disappears in one subsequent frame. Therefore, the similarity S is also defined as the reliability of block match. In the block match process of frame sequence, the reliability of every limb match is preserved. If there are one or several low reliability frames between two relatively high ones, we use the joint coordinates of high ones to obtain the joint coordinates of low ones by linear interpolation. Our experiment shows that it can reduce self-occlusion to a certain degree and optimize the tracking performance.

2.1.3 Reconstruction of 3D Human Motion Sequence

To establish the sequence of 3D human motion skeleton under the perspective projection, we must first acquire the camera parameter, i. e. camera calibration in computer vision. We use Newton method to solve this problem by the correspondences between 3D model and 2D image. Then we calculate the coordinates of the 3D feature points on the human model using the pinhole model and the proportion knowledge of human skeleton. In the frame sequence, the assumption of motion continuity is applied to eliminate the ambiguity of 3D motion information effectively.

2.1.3.1 Linear Model Based Camera Calibration

As shown in Fig. 2.6, consider two coordinate systems, $O_wX_wY_wZ_w$ and $O_cX_cY_cZ_c$. The former is an object space coordinate system in which the

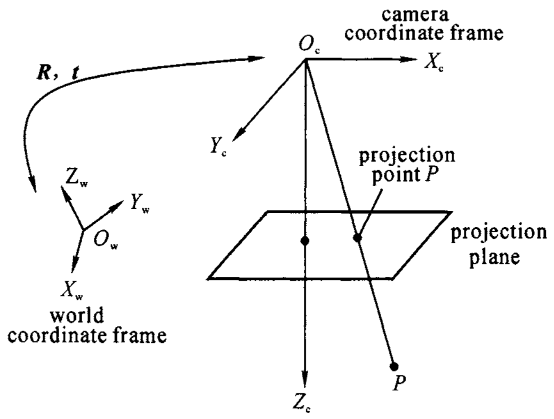


Fig. 2.6 Projective transformation

3D feature points are located. Thus P_w is a point in this coordinate system with coordinate (X_w, Y_w, Z_w). The camera is referenced to the camera coordinate system $O_cX_cY_cZ_c$. In particular, we assume that the image plane is perpendicular to the O_cZ_c axis and at location $Z_c = f$. Point coordinate on the image plane is obtained by the perspective projection and denoted by $P(u, v)$. Thus every point P_w in $O_wX_wY_wZ_w$ can be translated to (u, v) on the image plane with two transformations. Firstly, $O_cX_cY_cZ_c$ is obtained by a rotation $\boldsymbol{R}$ and translation $\boldsymbol{t}$ of the coordinate system $O_wX_wY_wZ_w$. The 3D coordinate of P_c is related to that of P_w by:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \boldsymbol{R}\left(\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} - \boldsymbol{t}\right) \tag{2-15}$$

Secondly, through the prospective projection, the projective point of P_c is at P whose coordinate is given by:

$$(u, v) = \left(\frac{f \cdot X_c}{Z_c}, \frac{f \cdot Y_c}{Z_c}\right) \tag{2-16}$$

The goal in camera calibration is to determine $\boldsymbol{R}$ and $\boldsymbol{t}$ when some corresponding features between 3D human model and 2D image plane are given. In the above two equations, it is difficult to calculate the partial derivative of u, v to unknown parameters. So we transform them into:

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \boldsymbol{R}\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} \tag{2-17}$$

$$(u, v) = \left(\frac{f \cdot X'}{Z' + D_z} + D_x, \frac{f \cdot Y'}{Z' + D_z} + D_y\right) \tag{2-18}$$

The meaning of $\boldsymbol{R}$ in the above equation is the same as that of equation (2-15). We substitute translation $\boldsymbol{t}$ with D_x, D_y, D_z. P' is a 3D point with coordinate (X', Y', Z'). These two representations are equivalent when $\boldsymbol{t}$ and D_x, D_y, D_z are related by:

$$\boldsymbol{t} = \boldsymbol{R}^{-1}\left[-\frac{D_x(Z + D_z)}{f}, -\frac{D_y(Z + D_z)}{f}, -D_z\right]^{\mathrm{T}} \tag{2-19}$$

The rotation parameter $\boldsymbol{R}$ can be represented by a rotation vector $(W_x, W_y, W_z)^{\mathrm{T}}$, whose direction is equal to that of rotative axis and whose module is equal to the rotation angle. Now the projective parameter can be represented by a vector, $[D_x, D_y, D_z, W_x, W_y, W_z]$ and the partial derivative of u, v to them can be calculated expediently. We use corresponding feature lines between 3D human model and 2D image plane to calculate projective parameters and define the equation of a line, with a point (u, v) on it, by:

$$\frac{-m}{\sqrt{m^2 + 1}}u + \frac{1}{\sqrt{m^2 + 1}}v = d \tag{2-20}$$

where d is the perpendicular distance from the origin to that line, and m is the line slope. From equation (2-20) we can get the partial derivative of d to u, v. Combining with previous calculation, the partial derivative of d to D_x, D_y, D_z, W_x, W_y, W_z will be obtained. After that, we may use Newton method to calculate a revision vector, $\boldsymbol{h}=[\Delta D_x, \Delta D_y, \Delta D_z, \Delta W_x, \Delta W_y, \Delta W_z]$.

This method is detailed as follows. Firstly, beginning with the initial values of projective parameters, $[D_x, D_y, D_z, W_x, W_y, W_z]$, let the 3D model project to the image plane according to the current parameters. Secondly, calculate the error between the projective line and the feature line on the image plane, which results in the following equation:

$$\frac{\partial d}{\partial D_x}\Delta D_x+\frac{\partial d}{\partial D_y}\Delta D_y+\frac{\partial d}{\partial D_z}\Delta D_z+\frac{\partial d}{\partial W_x}\Delta W_x+\frac{\partial d}{\partial W_y}\Delta W_y+\frac{\partial d}{\partial W_z}\Delta W_z=Ed \quad (2\text{-}21)$$

where Ed is the perpendicular distance from the end points of a 2D feature line to the projective line. Because there are two end points on one line, we can get two equations such as equation (2-21) for one pair of corresponding feature lines. So given three pairs of such lines, six equations will form a linear equation group. Its solving will lead to the revision vector $\boldsymbol{h}$. Then $\boldsymbol{h}$ is added to current projective parameters for revising projective parameters. Thus, we may solve the linear equation group again. All the Ed values in that equation group will be smaller than a predefined threshold after several iterations, meaning that the six projective parameters have been obtained.

There are at least three pairs of corresponding lines needed in Newton method. In the human model, we choose the line between left and right shoulders, and the two lines between the chest and two shoulders. These three lines constitute a steady isosceles triangle of human upper trunk. This choice is based on the observation that this triangle should not distort itself in human motion in most situations. In the following description, each feature object of this triangle is named as a *key joint*, *key line*, or *key triangle*. In the first frame, the projection of key joints on the image plane is known by manual marking. The key joint position in the object space coordinate is specified by our system. As long as the proportion of each key line accords with the anatomy, we can always find the location and orientation of camera in the object space coordinate system and let the perspective projection of key triangle superpose with the upper triangle of trunk on the image plane.

2.1.3.2 3D Reconstruction Corresponding to the First Frame

From the above, the six projective parameters have been obtained. Now corresponding to the first frame, except for three key joints, all the other 3D feature points of human model are not determined yet. The next step is to acquire the 3D feature point coordinate $P_c(X_c, Y_c, Z_c)$ of human model corresponding to a known 2D feature point coordinate $P(u,v)$. As known from the pinhole model, to link the optical center and a projected point will get a radial, on which all the points project on the same point in the image plane. In order to locate the 3D feature point on this radial, we begin with a known neighboring point p and use the knowledge of human skeleton length *Len* to find a point. The distance from the point to p is equal to *Len*.

Now we detail our algorithm by the example of inferring unknown right elbow point P_c from known neighboring point, the right shoulder point $P_c{}'$. The coordinate of right elbow point in the camera coordinate system is (X_c, Y_c, Z_c). (u,v) is the coordinate of P, in the image plane. There is an equation as follows:

$$d(P_c{}', P_c) = Len \tag{2-22}$$

where $d(P_c{}', P_c)$ represents the distance between $P_c{}'$ and P_c, *Len* is the upper limb length in the 3D human model. By combining equation (2-22) with (2-16), we can get an equation with only one variable Z_c. This equation can be visualized as a line intersected with a sphere with center $P_c{}'$ and radius *Len* (see Fig. 2.7). According to three possibilities, intersection, tangency and apartion, of a space line intersected with a sphere, the solution of this equation has also three cases:

- Two solutions. It means there are two possible positions for the elbow point. This ambiguity in the course of modeling from 2D to 3D is caused by this ill-posed problem itself. Two methods can be used to eliminate the ambiguity. Firstly, we can utilize diversified human anatomy constraints. For example, the lower arm cannot extend backward when the upper arm extends forward. Secondly, brightness information may be used. In the two solutions, one is always close to the optical center of camera and the other is far away. We make an

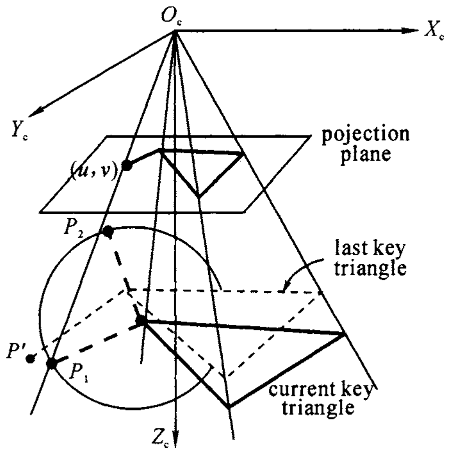


Fig. 2.7 Ambiguity elimination in the subsequent frames

assumption that an image region, which is closer to the optical center, has a relative higher brightness. We choose a small region around the feature point on the shoulder and the elbow respectively, transform the RGB color model to HLS model for every pixel in these two regions, and calculate the mean values of L component for two regions. If the value of elbow is larger than that of shoulder, we select the solution closer to the optical center, and otherwise we select the farther one. Our experiment shows that the combination of these two methods can eliminate the ambiguity effectively.

- One solution. We may get a unique point, which is just the position of 3D right elbow point.
- No solution. There are two reasons: one is the tracking error of 2D feature point, and the other is that the skeleton proportion of this person does not accord with the ordinary anatomy. The system will adjust the skeleton length for renewal computation.

Now, with the solution order from center to margin shown in Fig. 2. 8, we can get all the 3D feature point coordinates of the human model in turn.

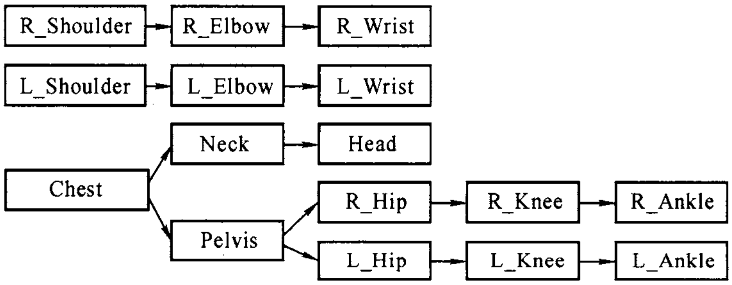


Fig. 2. 8 The solution order of 3D feature point

2. 1. 3. 3 3D Reconstruction in the Subsequent Frames

In the last subsection, we have constructed the 3D human motion skeleton for the first frame. In fact, as soon as the coordinates of three human key joints for every frame are known, the corresponding 3D human model can be obtained by the algorithm introduced in Sects. 3. 1 and 3. 2. Now we discuss how to determine the coordinates of three key joints in the subsequent frames.

Given the key joint coordinates, $P_i^n(X_i^n, Y_i^n, Z_i^n)$ $(i=1,2,3)$, of frame n in the camera coordinate system, let us calculate the corresponding key joint, $P_i^{n+1}(X_i^{n+1}, Y_i^{n+1}, Z_i^{n+1})$ $(i=1,2,3)$, of frame $n+1$. The corresponding 2D feature point in the image plane is (U_i^{n+1}, V_i^{n+1}). The relation of P_i^{n+1} and (U_i^{n+1}, V_i^{n+1}) can be described as:

$$P_i^{n+1} = \left(\frac{U_i^{n+1} \cdot Z_i^{n+1}}{f}, \frac{V_i^{n+1} \cdot Z_i^{n+1}}{f}, Z_i^{n+1}\right) \quad (i=1,2,3) \qquad (2\text{-}23)$$

As mentioned in Sect. 2. 1. 1, the skeleton length in the human model remains constant, which means:

$$d(P_i^n, P_j^n) = d(P_i^{n+1}, P_j^{n+1}) \quad (i, j = 1, 2, 3 \text{ and } i \neq j) \tag{2-24}$$

Using equation (2-23) to substitute P_i^{n+1} and P_j^{n+1} in equation (2-24), we will get a nonlinear equation group, which has three variables and may be solved by the grads method. Thus, we have obtained the key joint coordinate of frame $n+1$ in the camera coordinate system.

Then the algorithm mentioned in Sects. 3. 1 and 3. 2 is used to calculate all the 3D feature points in the human model corresponding to frame $n+1$. Here a key assumption is made: the human motion has the property of continuity. While the ambiguity appears, we calculate the distances of the two solutions to the 3D feature point of frame n respectively and select the solution with a smaller distance. Our experiment shows that this method has excellent performance. The continuity and authenticity are embodied in the long sequence of human motion. In the example of Fig. 2. 7, there are two possible positions, P_1 and P_2, while locating the right elbow point corresponding to frame $n+1$. From the dashed line in Fig. 2. 7, we know that the right elbow point in frame n is P'. Thus we select P_1 as the right elbow point of frame $n+1$ for it is closer to point P'.

2. 1. 4 VBHAS V1. 0

We implement a prototype system named Video-based Human Animation System (VBHAS V1. 0) using the algorithms introduced in this Chapter. The input of the system is an image sequence (or video clip) containing human motion and the output is the 3D coordinates of all joints in the skeleton. User labels the first frame manually, and the system interactively tracks the joints in subsequence frames. We don't put any constraints on the human motion, so sometimes the system tracks joints incorrectly. User interaction is incorporated to eliminate these errors. The 3D reconstruction is done by the system automatically. In the following we show some experimental results of VBHAS V1. 0.

2. 1. 4. 1 Sitting Motion

Figs. 2. 9 — 2. 14 show the results for a sitting motion at the 1st, 5th, 10th, 15th, 20th and 25th frames. The left image of each figure shows the 2D features on the frame image. The 16 features on the first frame are labeled by user with a mouse. Features on other frames are generated by VBHAS. It can be seen that the tracking results are excellent. The middle image of each figure shows the 3D reconstruction at the same viewpoint as the real frame and the right image shows the 3D reconstruction at a viewpoint rotated by 60 degree. The 3D reconstructions are also satisfactory. We can also note that the ambiguity is successfully eluded in each frame.

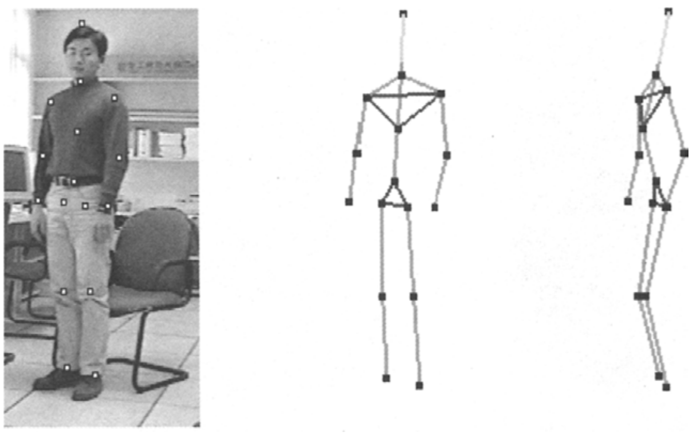


Fig. 2. 9 Sitting motion at the 1st frame

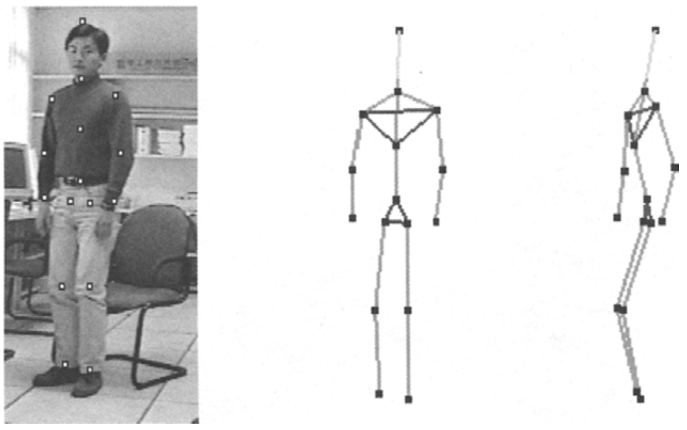


Fig. 2. 10 Sitting motion at the 5th frame

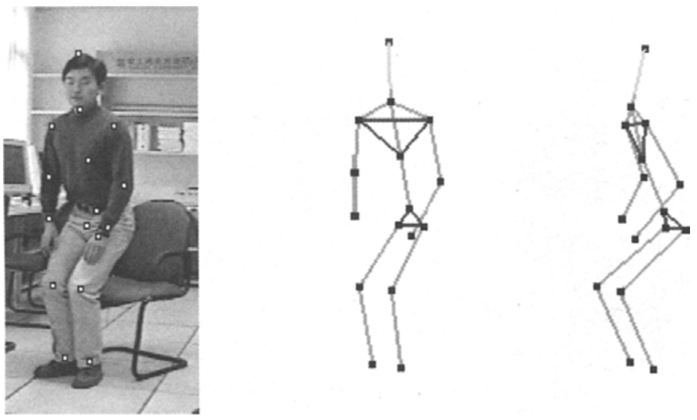


Fig. 2. 11 Sitting motion at the 10th frame

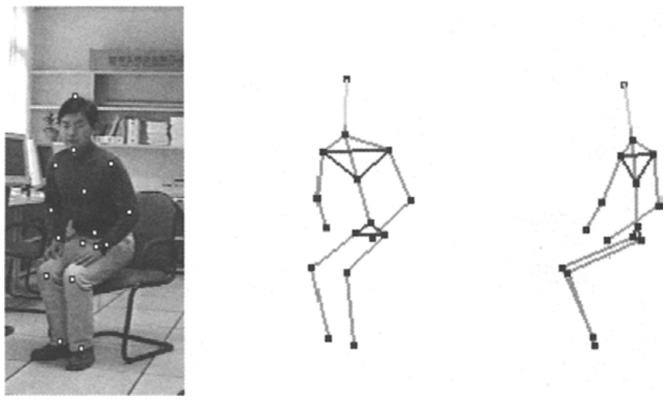


Fig. 2. 12 Sitting motion at the 15th frame

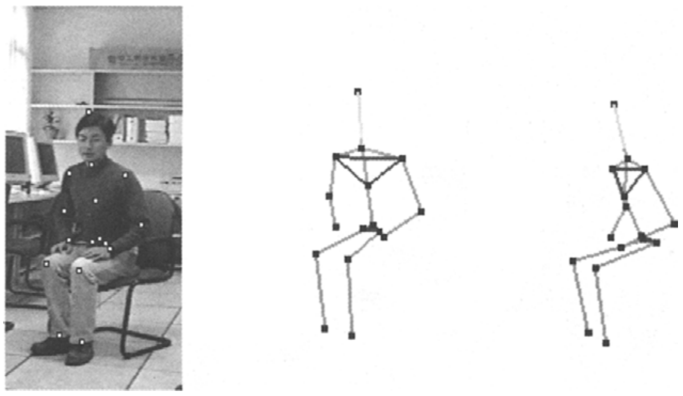


Fig. 2. 13 Sitting motion at the 20th frame

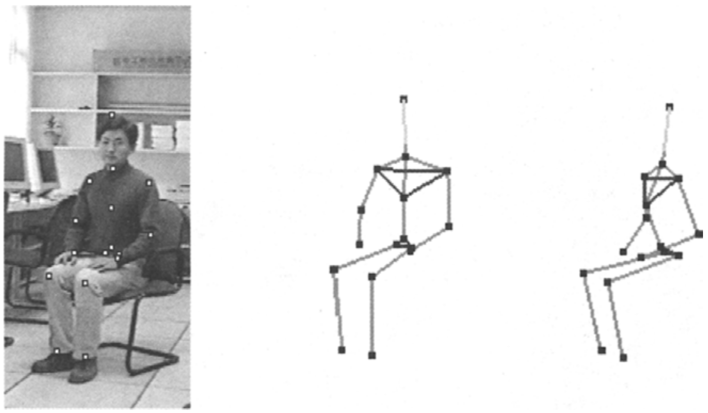


Fig. 2. 14 Sitting motion at the 25th frame

2.1.4.2 Walking Motion

Figs. 2.15—2.19 show the results for a walking motion at the 1st, 5th, 10th, 15th and 20th frames. In this case, the projected human height varies notably with time. The results show that the algorithm can deal with this situation.

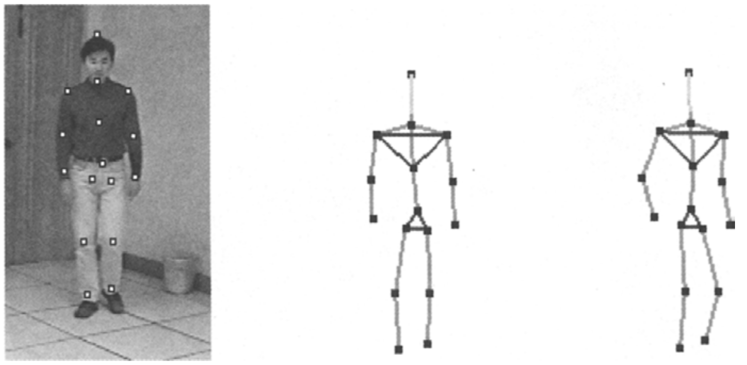


Fig. 2.15 Walking motion, at the 1st frame

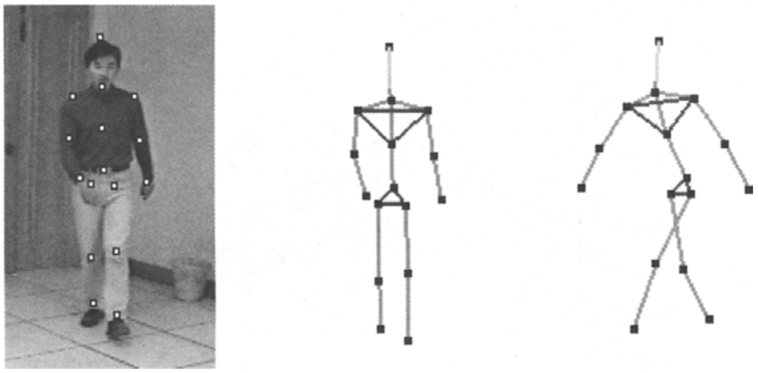


Fig. 2.16 Walking motion at the 5th frame

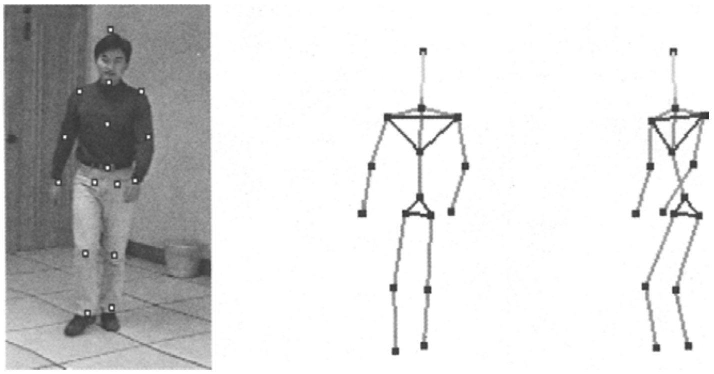


Fig. 2.17 Walking motion at the 10th frame

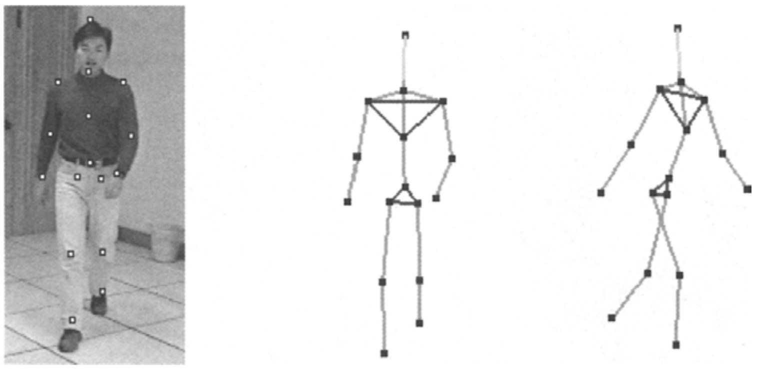


Fig. 2. 18 Walking motion at the 15th frame

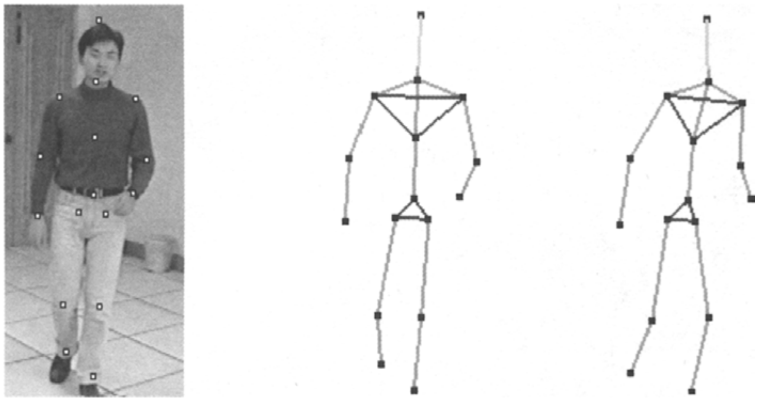


Fig. 2. 19 Walking motion at the 20th frame

2.1.4.3 Movie Mask

We extract a clip of 2 seconds from the movie *Mask*. The clip contains 48 frames. Figs. 2. 20—2. 24 show the results for this clip at the 1st, 5th, 17th, 26th and 39th frames. It is well known to the audience of that movie that in this clip the dancing is performed at a very high speed with significant displacement of limbs. Our system can successfully track such fast motion.

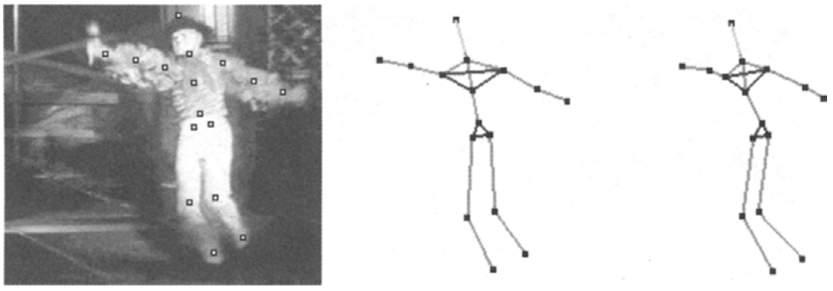


Fig. 2. 20 Clip from Mask at the 1st frame

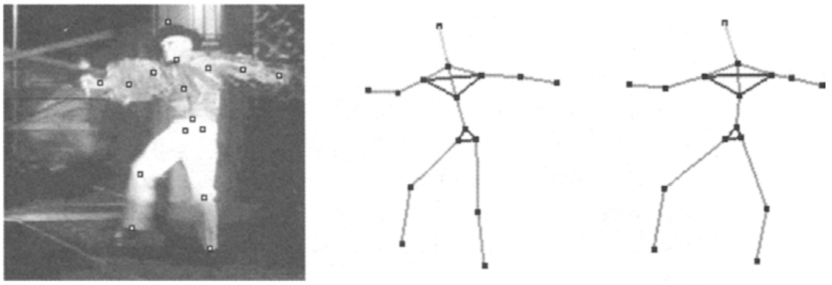


Fig. 2. 21 Clip from Mask at the 5th frame

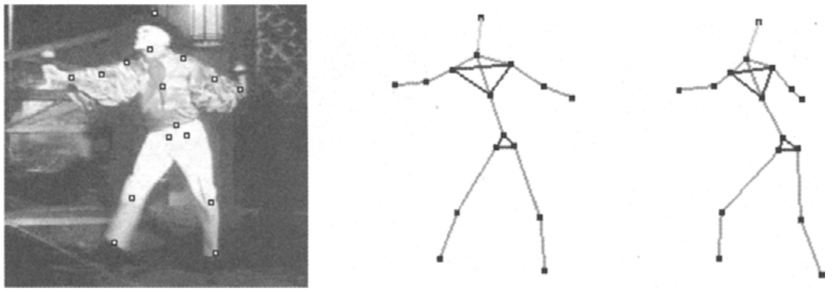


Fig. 2. 22 Clip from Mask at the 17th frame

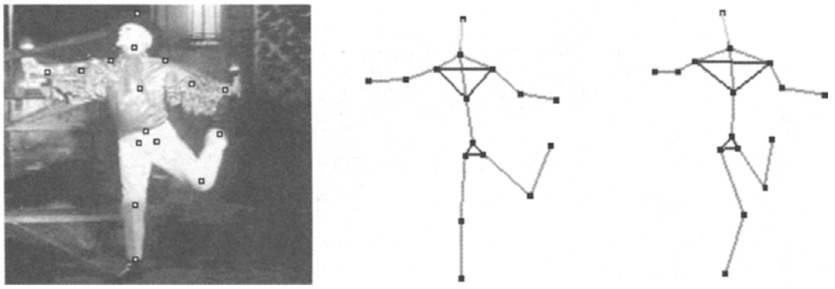


Fig. 2. 23 Clip from Mask at the 26th frame

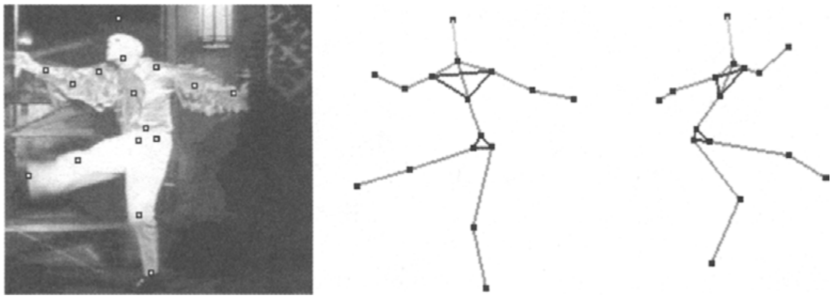


Fig. 2. 24 Clip from Mask at the 39th frame

2.1.5 Discussions

Above we have discussed our algorithm in detail for 3D human motion capture based on feature tracking. Our system VBHAS V1.0 can successfully track and reconstruct many human motions. But there are still some limitations which will be discussed below.

First let's discuss some limitations in the 2D feature tracking.

- If the colors of clothes and background are very similar, the tracking will be affected. Because in such cases there is no obvious boundary between foreground and background, the matching of image blocks will be heavily interfered.
- When part of a limb has very large deformation at some instant, the tracking will also be degraded. In other words, the system deals with gradual deformation better.
- The skeleton model we proposed cannot accommodate some degrees of freedom of real human motion.
- If frequent self-occlusion is present, the tracking performance will degrade heavily. We propose two methods to solve this problem. First, user interaction can be incorporated. Second, 3D prior knowledge of human body can be employed to minimize user interaction.

Next we discuss some limitations in the 3D reconstruction.

- ■ If error occurs in 2D feature tracking, then the 3D reconstruction will be impaired, since the 3D reconstruction is built entirely upon the 2D feature tracking. And in many cases the 3D reconstruction can even magnify errors.
- ■ If the body model of the real subject is notably different from the standard model, the 3D reconstruction can also be affected. The solution is to adjust the model used by the algorithm, either automatically from the video, or manually by user.

2.2 Human Motion Capture Based on Silhouette

2.2.1 Overview

In this section we present a robust framework for recovering 3D human motions from uncalibrated monocular videos (or image sequences) without markers. This framework first extracts silhouettes from the images and then analyzes the silhouettes to get the positional information of 2D body configurations. The silhouettes and the derived information then serve as the sources of an object function we try to minimize. The function is computationally efficient and reflects the correspondence between a 3D pose and the silhouette. Finally, smooth motions are generated from the recovered poses. This framework is free from common problems in motion tracking such as error propagation and difficulty in recovering from tracking errors. Moreover, the system can locate where user should specify manual frames in order to complete the recovery robustly, which is especially important to minimize user intervention. The experimental results show that even in difficult situations, complex motions of a large variety of types can be robustly recovered with reasonable intuitive user interactions.

Fig. 2. 25 shows the system workflow. First, silhouettes are extracted from video and are analyzed to determine the 2D positions of body trunk and head/hands/feet. Then, 3D poses are recovered from the analyzed silhouettes. Finally, a smooth motion is generated from poses.

This section is organized as follows. First we introduce silhouette extraction and silhouette analysis. Then the object function and optimization process to recover poses are elaborated. After that, the strategy to recover an entire motion clip is provided. At last we present the experimental results and discussions.

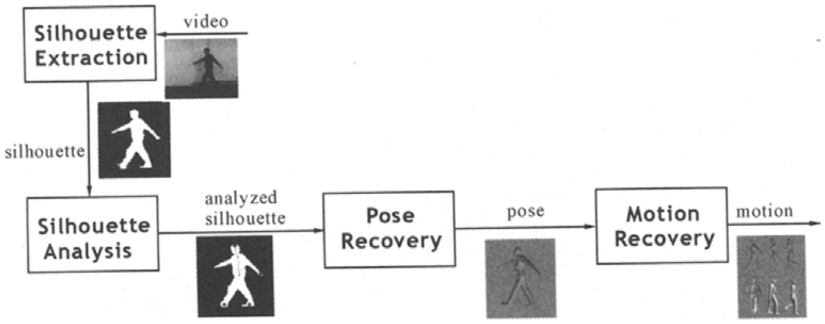


Fig. 2. 25 System workflow

2. 2. 2 Silhouette Extraction and Analysis

The first step is extracting silhouettes from video and removing noises. We choose to recover poses via silhouettes as the following reasons:

- Silhouettes can be relatively robustly extracted, especially when we have access to the background model;
- Silhouettes are relatively insensitive to noises and irrelevant factors such as the clothes texture or the dynamic cockles, which sometimes make the color, texture or internal edges caused tracking fail;
- Simple as they look, silhouettes preserve a lot of information.

Silhouette extraction involves segmenting images into foreground and background. Current segmenting methods are based on the optical flow [10], temporal image differentiation [11], or background subtraction[12]. Here we use a simplified method by assuming static background and static camera. Horprasert, et al.[12] employed a statistical approach to model the background where a color model of brightness distortion and chromaticity distortion helps remove shadows. The method is robust against shadows and other global or local illumination changes. We take a similar approach as theirs. Other more complex methods could also be used. For example, Haritaoglu, et al.[13] presented a method that uses a dynamic background model to accommodate background changes.

The extracted silhouettes are processed by a pass of erode and dilate. Then noises are further reduced by removing isolated foreground pixels and filling tiny holes.

We denote the silhouette from video at frame t to be $\boldsymbol{Sv}^t$ ($t=1,2,\cdots,T$, where T is the total frame number). $\boldsymbol{Sv}^t$ is actually a binary function:

$$\boldsymbol{Sv}^t(x,y)=\begin{cases}1, & \text{if pixel on row } x \text{ and column } y \text{ is foreground}\\ 0, & \text{otherwise}\end{cases} \tag{2-25}$$

After silhouette extraction, the next step is to get some 2D geometric information. We try to determine the 2D positions of body trunk and hands/feet on the silhouette.

By "body trunk" we mean the main axis of the torso between the centroid and the head. It should be noted that sometimes arms are raised to a position higher than head, so body trunk cannot be simply determined by the centroid and the highest location in the silhouette. Vignola, et al.[14] proposed a method that can deal with this problem, but their method approximates the trunk with a straight line and may fail when the trunk bends. We extend their method. First, the Euclidean distance transform on the silhouette is calculated, which, for every point, finds its distance to the nearest boundary. Then for every horizontal line above the silhouette's centroid, we find a point with the largest distance value. In this way, we get a curve of the most "inward" points on every horizontal line. If hands are not raised higher than head, the curve will be continuous, going from centroid all the way up to the head's top ((a) and (b) in Fig. 2. 26). If the hands, however, are raised higher than head, discontinuity will happen in the curve, and the discontinuity point is at the head. In such cases, we cut the curve at the discontinuity point and only reserve the lower part ((c) and (d) in Fig. 2. 26). After that, we fit a 2nd order polynomial on the curve as the body trunk. The 2nd polynomial approximates the trunk quite well because it allows smooth and gradual bending, while avoiding complex winding such as S shape.

Next, we detect hands and feet. Perhaps the most intuitive way is by looking at the curvatures on the silhouette's boundary. Hands and feet are places where large positive curvatures are encountered. The detailed process is as below.

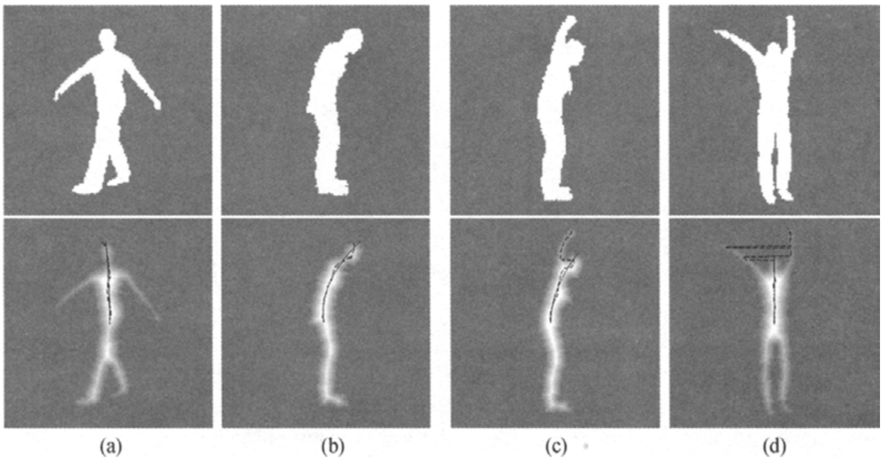


Fig. 2. 26 Trunk discovery. The points with largest distance value on every horizontal line form the dash curve, from which we fit a 2nd polynomial (the solid curve) as the trunk. If discontinuity occurs on the dash curve, then only the segment below the discontinuity point is used

Step 1 Forevery point on the silhouette's boundary, calculate its curvature. The curvature is approximated by the angle made by a point and its two neighbors n points away, where n is a small number. This helps to exclude abrupt curvature changes caused by noises.

Step 2 Mark out all points whose curvatures are larger than a threshold.

Step 3 Conduct a clustering on all marked points along the boundary. Usually the marked points are already well clustered, so very simple clustering method is sufficient. Clusters with too few points are discarded.

Step 4 All the remaining clusters are hands or feet.

Note that if a hand or foot is not stretched and cannot be distinguished from the torso on the silhouette, it cannot be detected.

Some other methods could also be used to detect hands/feet. We can simply construct a kernel that resembles hands or feet and convolve it with the silhouette. Another method proposed by Fujiyoshi and Lipton[15] uses silhouette's star skeleton to analyze human motion. The distances between silhouette centroid and boundary points are calculated and form a one-dimensional function, and obvious local maxima are taken as head, hands or feet. We implement the abovementioned methods and it is somewhat surprising that in most cases the curvature method works best. Different methods can be used jointly to get higher accuracy.

So far we have only got the positions of hands/feet but we don't know which one is the left hand or the right foot, so we have to label them. The automatic labeling is based on temporal information. User is required to manually label a frame where all hands and feet are detected. Then the system automatically labels other frames successively. In order to achieve higher accuracy, we employ Kalman filter to make predictions before labeling. Movements of hands/feet are assumed as with constant acceleration locally.

Fig. 2. 27 shows some results of silhouette analysis. Note that the head is always detectable along with the trunk, but hands/feet might not be detected for some frames. We use four binaries π^t_{lh}, π^t_{rh}, π^t_{lf}, π^t_{rf} to denote if corresponding hand or foot is detected on silhouette at frame t (1 if detected and 0 otherwise; subscripts lh and rf for left hand and right foot, and so on); and we denote the detected positions of head, hands and feet on frame t as $\boldsymbol{l}^t_{\mathrm{head}}$, $\boldsymbol{l}^t_{\mathrm{lh}}$, $\boldsymbol{l}^t_{\mathrm{rh}}$, $\boldsymbol{l}^t_{\mathrm{lf}}$, $\boldsymbol{l}^t_{\mathrm{rf}}$ (if a position is not detected, then corresponding $\boldsymbol{l}$ is not defined).

The information derived in silhouette analysis is used in pose recovery later. The trunk is used for scaling and aligning skeleton to the silhouette; the end sites' locations are used in the object function, as we elaborate below.

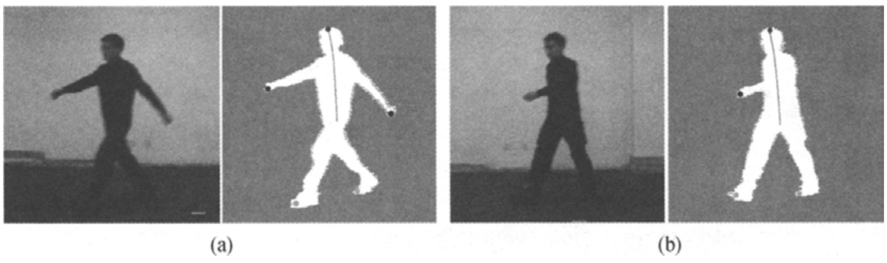


Fig. 2. 27 Silhouette analysis result. Note in the second silhouette only one hand is detected

2.2.3 Pose Recovery

We consider the pose recovery as an optimization problem: find a pose vector $\boldsymbol{p}^t$ at frame t that optimizes a scalar object function $E(\boldsymbol{p}^t)$ that encodes the dissimilarity between the pose and the silhouette.

First we formulize the pose vector $\boldsymbol{p}^t$. Each joint in the human skeleton model has 3 DOFs which quantify its rotation in its parent's local coordinate frame. Due to biomechanical constraints, each DOF has its own valid range. The root joint has another 3 DOFs, namely, its translation in the world coordinate. In pose recovery step, we consider pose as the vector of all joints' rotations including root. The root's translation is not included since it is the figure's global position instead of pose configuration.

Next we formulize the object function $E(\boldsymbol{p}^t)$. $E(\boldsymbol{p}^t)$ must reflect the dissimilarity between the pose and the silhouette, and it should be computationally efficient, since the object function will be evaluated many times in the optimization process. We design an object function $E(\boldsymbol{p}^t)$ that has the following three components.

The core-area term

When the human skeleton model is configured at the same pose as indicated in silhouette, the body trunk and four limbs on the skeleton, after scaling and aligning, should lie in the core areas (or medial axes) of the silhouette. To highlight the core areas, silhouette's Euclidean distance transform is used. See Fig. 2. 28. To quantify this core-area constraint, we uniformly sample some points on the trunk and limbs on the skeleton model, calculate the 2D positions of these points when the skeleton is configured at pose $\boldsymbol{p}^t$, and then define $E_{\text{core-area}}$ as the negative average distance value at the sampled points' 2D locations on the silhouette's Euclidean distance transform.

$$E_{\text{core-area}} = -\frac{1}{M}\sum_{m=1}^{M} \| \boldsymbol{Sv}_{\text{dt}}^t(x_m, y_m) \| \tag{2-26}$$

where M is the total count of sampled points, (x_m, y_m) is the calculated 2D position of the mth point, and $\boldsymbol{Sv}_{\text{dt}}^t$, which is the distance transform of silhouette $\boldsymbol{Sv}^t$, takes in a position coordinate and returns the distance value to the nearest edge.

Note that the scaling and aligning of skeleton against the silhouette are conducted using the detected trunk on the silhouette. Besides, if some end sites are not detected on the silhouette, the corresponding limb does not contribute to $E_{\text{core-area}}$.

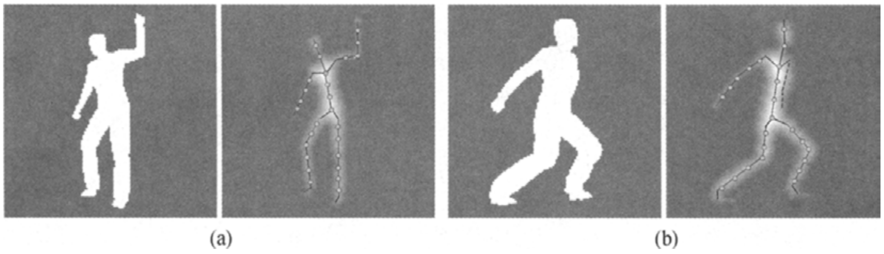


Fig. 2. 28 The definition of $E_{\text{core-area}}$. The dashed part of the skeleton is the trunk and limbs where we sample points, which are represented by circles. This is only an illustration; real sampled points are denser. Note in silhouette (b) the dashed arm is occluded and no points are sampled on it

The coverage term

It is not sufficient to determine a pose merely by the core-area term. $E_{\text{core-area}}$ puts constraint in one direction. It says that trunk and limbs should lie in the core areas of silhouette, but it is not required that each core area of the silhouette should be fully covered by some limb. It is possible that an arm is not stretched enough to fill in the entire arm's core area, or even the arm might be mistakenly superimposed on the torso, while a very good

$E_{\text{core-area}}$ value is still maintained. Therefore, we need another constraint E_{coverage} that controls the coverage of the limbs. In silhouette analysis we have determined head/hands/feet positions on the silhouette. And we can use their positions to estimate the limbs' coverage. We define E_{coverage} as the mean distance between the 2D positions of head/hands/feet on the skeleton model and the detected counterparts on the silhouette.

$$E_{\text{coverage}} = \| \boldsymbol{l}_{\text{head}} - \boldsymbol{l}'_{\text{head}} \| + \pi_{\text{lh}} \| \boldsymbol{l}_{\text{lh}} - \boldsymbol{l}'_{\text{lh}} \| + \pi_{\text{rh}} \| \boldsymbol{l}_{\text{rh}} - \boldsymbol{l}'_{\text{rh}} \| + \pi_{\text{lf}} \| \boldsymbol{l}_{\text{lf}} - \boldsymbol{l}'_{\text{lf}} \| + \pi_{\text{rf}} \| \boldsymbol{l}_{\text{rf}} - \boldsymbol{l}'_{\text{rf}} \| \tag{2-27}$$

where $\boldsymbol{l}$ is the position detected on the silhouette and $\boldsymbol{l}'$ is the position calculated by projecting skeleton to 2D plane. Here we have dropped all superscript t for simplicity.

Note that if an end site is not detected on the silhouette, it simply does not make contribution to this term.

The smoothness term

Finally, the smoothness term guarantees that the poses do not change abruptly temporally. We use two previous frames to measure the smoothness.

$$E_{\text{smoothness}} = \| \boldsymbol{p}^t - 2\boldsymbol{p}^{t-1} + \boldsymbol{p}^{t-2} \|^2 \tag{2-28}$$

Combining the three terms, the object function is:

$$E(\boldsymbol{p}^t) = a_1 E_{\text{core-area}} + a_2 E_{\text{coverage}} + a_3 E_{\text{smoothness}} \tag{2-29}$$

Derivation-based optimizing methods cannot be used to minimize $E(\boldsymbol{p}^t)$. Also, the complexity of $E(\boldsymbol{p}^t)$ requires the optimizing method be able to avoid being trapped in local minima. We adopt Simulated Annealing (SA), which converges to the global minimum with probability 1 when the initial temperature is sufficiently high and the annealing process is sufficiently slow. In practice we cannot ensure these two "sufficiency" but SA does provide great help to get rid of the local minima. Also, by setting the parameters such as the initial temperature and annealing rate, it is very intuitive to control how much ability we give SA to get out of a local minimum. Also note that when the initial temperature is high, the initial configuration is of little importance, which helps to eliminate the need of providing a first guess in some cases, giving chances to get rid of error propagation/accumulation.

2.2.4 Motion Recovery

We have described how to recover a pose from a frame. When recovering motions from video, it is not desirable to recover the poses from frames separately and simply combine the poses into motion sequence. It is not possible to evaluate the 3D configuration of a limb if it is not identified on the frame. On the other hand, as illustrated by [16], recovering a pose from a single frame may have reflective ambiguity that is difficult to deal with. Therefore, temporal information should be exploited when recovering motions.

The detection of end sites is a very important source in our pose recovery process and we can imagine that it is easier to recover pose from a frame where all limbs are stretched out and detected, even little temporal cue is used. We classify all the frames into two types. Frames where all the end sites are identified are classified as *reliable frames*, while frames where at least one hand or foot is not identified are classified as *unreliable frames*. Obviously, reliable frames and unreliable frames tend to cluster, dividing the whole video into interleaved *reliable sections* and *unreliable sections*. Fig. 2. 29 shows the segmentation of a real video clip.

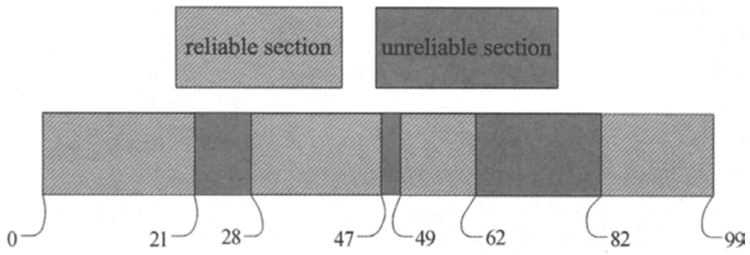


Fig. 2. 29 Segmentation of a video clip into reliable and unreliable sections. The numbers on boundaries are the last frame numbers in the closing sections

Clearly, reliable frames have larger chances to be correctly recovered. Also, since temporal cue can be employed, unreliable frames near the section boundaries (such as the 64th frame in Fig. 2. 29) have larger hope than the ones deep inside the section (such as the 70th frame) to be correctly recovered. We set a threshold L, if an unreliable section is longer than L, the frames deep inside it have so poor recovery chances that user is required to specify a manual frame in the middle of the section. When specifying a manual frame, user is asked to specify the positions of all the joints on limbs. The manual frame splits the long unreliable section into two short ones and this procedure may be repeated until no unreliable sections longer than L is present. Then the following recovery algorithm is taken.

Step 1 For each reliable section, automatically select a base frame. The selection is directed by intuition that frames where the four limbs are stretched far apart are easier to recover, so a frame with the maximum distance among the hands and feet is selected.

Step 2 Recover base frames, using full SA but not temporal information (a_3 in equation (2-29) set to 0). Full SA means high initial temperature and no initial guess (initial pose is set to be $\mathbf{0}$).

Step 3 For each base frame, incrementally recover its surrounding frames both forward and backward, using temporal information and partial SA, until all frames in two surrounding unreliable sections are covered. For example, if the 40th frame in Fig. 2. 29 is selected as a base frame, we incrementally recover the 21st frame to the 49th frame from it. Partial SA means low temperature and with initial guess by the surrounding recovered poses.

Step 4 Now for every unreliable frame, two pose candidates are obtained from the two surrounding base frames. We blend the two candidates to get the result, using the inverses of the distances to the two surrounding section boundaries as the weights.

In Step 2, base frames are recovered independently, without any temporal information. This means that our algorithm self-restarts at every base frame and so is free from error propagation. Even if some frames in an unreliable section are recovered completely wrong, the error will not pass over the section boundary. Besides, since the end sites are labeled as left or right in silhouette analysis and temporal information is used for unreliable frames, reflective ambiguity is gracefully avoided.

Now we have pose candidates for all frames, from which smooth motion is generated. We combine the pose candidates, and apply a Gaussian low pass filter to ensure smoothness.

There remains one important thing we have overlooked so far. We discarded the feet's translation during pose recovery, and now we have to recover them in order to get the correct motion. This task is straightforward if the motion is known to be ground-contact static, e. g. at any time at least one foot is on the ground and no slide exists, because we can always tell which foot is on the ground via the recovered 3D pose. For motions in which the body may fly in the air (such as jumping) or slide occurs (such as skiing), feet's locations are generated by key-frame, where user is required to specify the feet's location for some key-frames.

2. 2. 5 Results

We use a human skeleton model with 16 joints including feet. Therefore, the pose vector is 48D. Note that due to biomechanical limitation, the valid range for pose vector is a closed sub-region in 48D Euclidean space.

We test our system on some types of motions. Fig. 2. 30 shows the recovery result for a synthetic video of jumping followed by a turning. The video has about 400 frames. This scenario is not particularly difficult, since during the jumping the body is towards the camera and hands/feet are generally stretched out. Since we keep track of the relation of left and right, the turning is also easy to follow.

Fig. 2. 31 shows the result on a real walking video. Our system successfully recovers the 3D motion with 110 frames.

Fig. 2. 32 shows the result for a real shadowboxing video of about 1,100

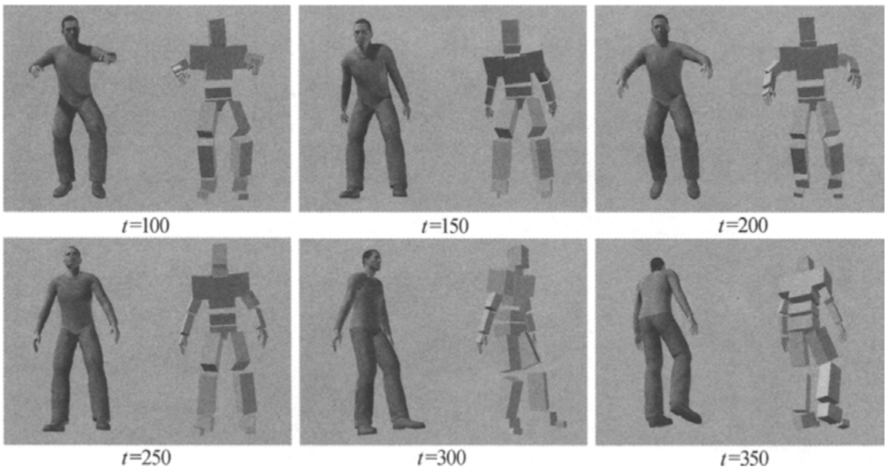


Fig. 2.30 Results for synthetic jumping video

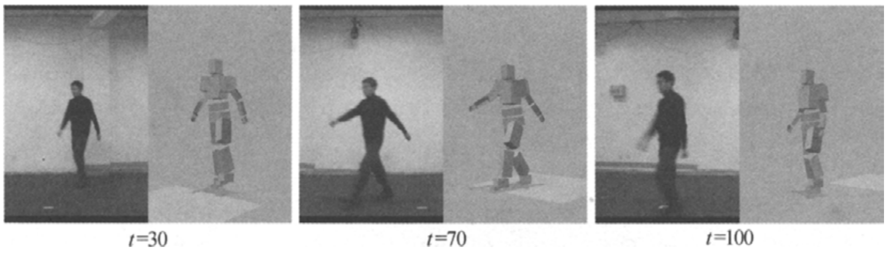


Fig. 2.31 Results for real walking video

frames. This is the most challenging one in our experiments. The difficulty comes from the loose clothes which make the body's trunk and limbs significantly thicker than the standard human model, as well as the motion complexity and the video length. Here we can see one of the advantages of our method. The fitting of a skeleton instead of a full body model [17] against a distance transformed silhouette greatly alleviates the problem of model dissimilarity. However, our system fails to follow a turning in the middle of the motion, so user has to manually specify the yaw angle for several frames as constraints.

2.2.6 Discussions

In this section we have described a system that robustly reconstructs 3D human motions from monocular videos. Instead of tracking feature points directly on the image, which is very sensitive to noises and self-occlusions, we take a model-based method using silhouettes, as in many applications silhouettes can be relatively reliably extracted. Our system first analyzes the silhouette to get the 2D positions of body trunk and end sites (head, hands and feet). Then, poses are recovered. For reliable frames, less tem-

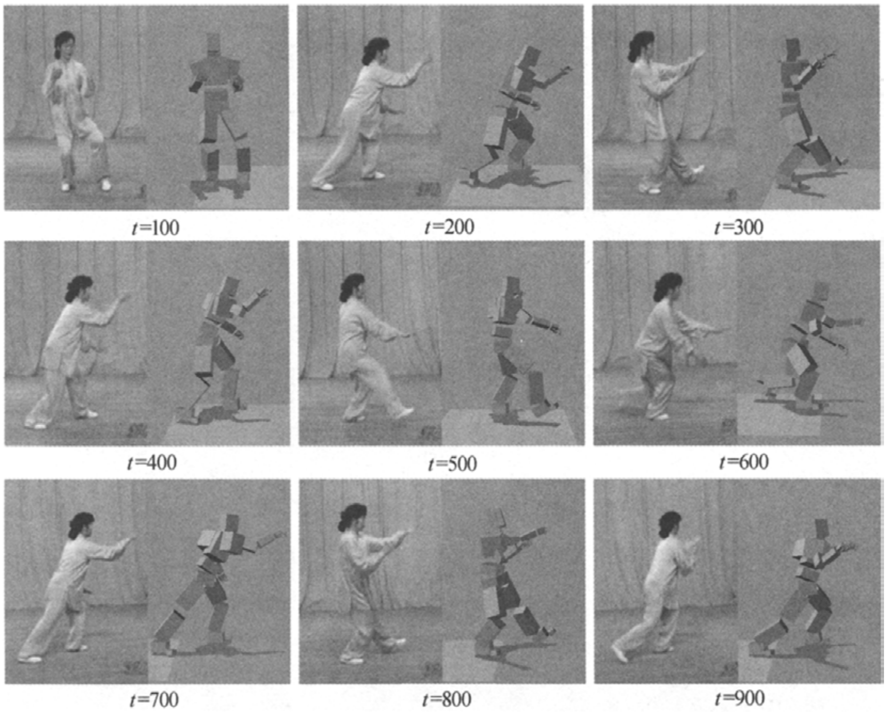


Fig. 2. 32 Results for real shadowboxing video

poral information is used, while for unreliable frames, temporal information plays an important role. Also, if some unreliable sections are too long to be easily recovered, manual frames specified by user are exploited.

We fit the skeleton instead of full body model [17] to the silhouette to help alleviate the effect caused by irrelevant model dissimilarity. This also significantly reduces the computational burden of the object function and allows the use of computational intense simulated annealing to help avoiding local minima. In this way, we gracefully tackle the common two difficulties in model-based methods presented in the introduction.

Our system successfully recovers different kinds of full body motions with reasonable user interaction. Compared to other systems, ours has the following advantages.

- More robust. We don't build the pose recovery on feature points tracking. And our system doesn't suffer from error propagation.
- Occlusions are dealt with gracefully. Short occlusions can be resolved automatically, while long occlusions, which are indeed very difficult to recover, are recovered by user specified information.
- The system has the ability to locate places where reconstruction is difficult and user interaction is needed. This is the key to minimizing us-

er intervention. It is improper to let the user flip through the frames and make the decision.

- The system is highly adaptive. By setting a few parameters such as threshold L, it is very easy to meet different requirements. For example, when accuracy requirement is very high, it is straightforward to tune down L, which will cause potentially more manual frames but a more accurate result. Also, SA is inherently highly adaptive by intuitively setting its parameters.

The inspiration of the system is from the observation that current techniques for 3D reconstruction from monocular video remain far from satisfaction. Some systems claim the ability of fully automatic reconstruction of unconstrained human motion, but the robustness problem prevents their practical use. Other systems are more robust, but they only work for particular motion types and need proper databases and extensive training. We feel that at the current technical level, a more practical and useful system might be one that can robustly recover motions without prior knowledge, but with some user interaction when necessary. Also, to minimize user interaction, the system should be able to determine where user interaction is mostly needed. Unfortunately, there seems to be not much of this in literatures.

Video-based reconstruction of complex human motions is always a challenging work, and our system does have some limitations. In videos where highly clustered background and unpredictable camera movement are present, silhouette extraction is a difficult task; and now we are studying this problem. Also, since temporal information is employed, our system might fail on abrupt or very fast motions because of the sample rate of plain videos. Cameras with high sampling rates can solve this problem, but this is also a serious limitation to the system's usage.

References

1. O'Rourke J, Badler NI. Model-based image analysis of human motion using constraint propagation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2(6): 522-536, 1980.
2. Hogg D. A program to see a walking person. Image Vision Computing, 1(1): 5-20, 1983.
3. Rohr K. Incremental recognition of pedestrians from image sequences. In: Computer Vision and Pattern Recognition, pp. 8-13, IEEE Computer Society, 1993.
4. Chen Z, Lee HJ. Knowledge-guided visual perception of 3D human gait from a single image sequence. IEEE Transactions on Systems, Man, and Cybernetics, 22(2): 336-342, 1992.

5. Bharatkumar AG, Daigle KE, Pandy MG, Cai Q, Aggarwal JK. Low limb kinematics of human walking with the medial axis transformation. In: Workshop on Motion of Non-Rigid and Articulated Objects, pp. 70-76, IEEE Computer Society, 1994.
6. Akita K. Image sequence analysis of real world human motion. Pattern Recognition, 17(1): 73-83, 1984.
7. Bregler C, Malik J. Video motion capture. UC Berkeley, Technical Report CSD-97-973. http://www, cs. berkeley. edu/~ bregler/bregler-malik-sig98. ps. gz.
8. Aggarwal JK, Cai Q. Human motion analysis: a review. Computer Vision and Image Understanding, 73(3): 428-440, 1999.
9. Liu M, Yao H, Gao W. Real-time human face tracking in color images. Chinese Journal of Computer, 21(6): 527-532, 1998.
10. Kearney JK, Thompson WB. Bounding constraint propagation for optical flow estimation. In: Aggarwal JK and Martin W, ed. Motion Understanding, Kluwer, 1988.
11. Abdel-Malek A, Hasekioglu O, Bloomer J. Image segmentation via motion vector estimates. In: Medical Image IV: Image Processing, pp. 366-371, SPIE, 1990.
12. Horprasert T, Harwood D, Davis L. A statistical approach for real-time robust background subtraction and shadow detection. In: ICCV Frame-Rate Workshop, IEEE Computer Society, 1999.
13. Haritaoglu I, Harwood D, Davis LS. W4: who? when? where? what? A real time system for detecting and tracking people. In: FGR'98, pp. 222-227, IEEE Computer Society, 1998.
14. Vignola J, Lalonde J, Bergevin R. Progressive human skeleton fitting. In: Proc of 16th Conference on Vision Interface, pp. 35-42, 2003.
15. Fujiyoshi H, Lipton A. Real-time human motion analysis by image skeletonisation. In: Proc of the Workshop on Application of Computer Vision, pp. 15-21, IEEE Computer Society, 1998.
16. Agarwal A, Triggs B. Recovering 3D human pose from monocular images. IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(1): 44-58, 2006.
17. Chen Y, Lee J, Parent R, Machiraju R. Markerless monocular motion capture using image features and physical constraints. In: Computer Graphics International, pp. 36-43, 2005.

3

Human Motion Capture Using Color Markers

At the current technical level, capturing human motion data accurately and directly from movie clips remains to be a difficult problem. In the scenarios with frequent self-occlusion, the performance of VBHAS V1. 0 will be impaired, which limits the system's use in some professional fields requiring high accuracy (e. g. sports analysis, mathematical analysis).

To overcome this difficulty, we design a suit of tight clothing attached with color markers and the technique is also developed which can track unconstrained human motion using this tight clothing. This chapter will discuss in detail the technique and the corresponding system VBHAS V2. 0.

3. 1 Tracking Color Markers

There exists some previous work on 2D tracking using color markers. Rehg and Kanade [1] used self-adaptive template method in the hand tracking. Hogg [2] developed a system based on feature tracking aiming at walking motion with only one DOF. Segen and Pingali [3] used corner points of moving edges as features. Bregler and Malik [4] used a method of solving simple linear system. In [5—7], Sum of Squared Distance (SSD) technique was employed. However, these methods do not perform well with complex motions. The results of tracking color markers are the 2D positions of the marker's centers in the image. Fig. 3. 1 shows the workflow of the tracking using color markers. The method can estimate the centers of color markers accurately and validate the tracking results automatically. If the validation fails, the system takes some other steps (morphing block matching) to ensure correct tracking.

This section is organized as follows. First the adopted human model and color space are presented. Then we illustrate the use of Kalman filter to predict for the next frame. After that, we briefly introduce the edge de-

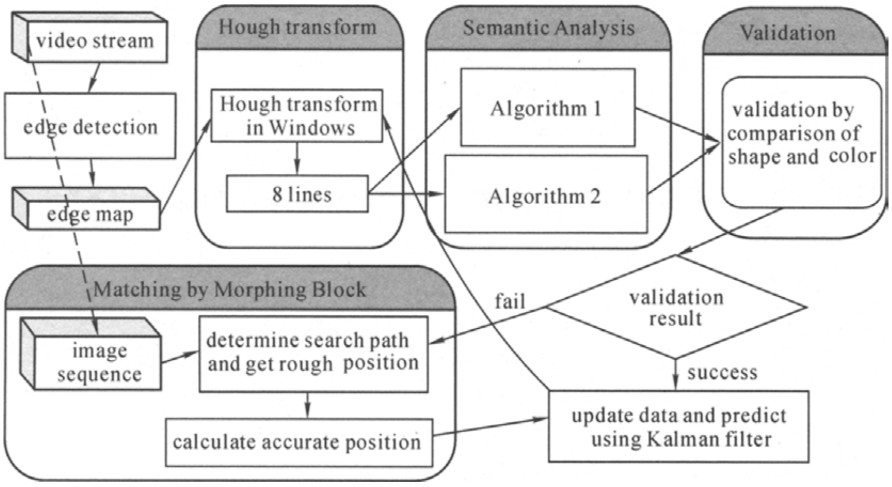

video stream
edge detection
edge map
Hough transform
Hough transform in Windows
8 lines
Semantic Analysis
Algorithm 1
Algorithm 2
Validation
validation by comparison of shape and color
validation result
fail
success
Matching by Morphing Block
image sequence
determine search path and get rough position
calculate accurate position
update data and predict using Kalman filter

Fig. 3. 1 System overview

tection algorithm and line extraction algorithm. Then, two algorithms to construct optimal rectangles from lines are elaborated. Finally, a more robust tracking algorithm based on morphing block matching is presented.

3. 1. 1 Human Model and Color Space

We view the 3D human body as a rigid articulated model (Fig. 3. 2). For example, an upper limb is formed by an upper arm and a lower arm linked by the elbow, and the upper limbs and torso are linked by shoulders. There are 16 joints in the human model. In our tight clothing, 15 color markers (corresponding to 15 joints in the human model excluding head) are attached to the gray clothes. The head joint can be treated as a marker without the use of color markers on the clothes.

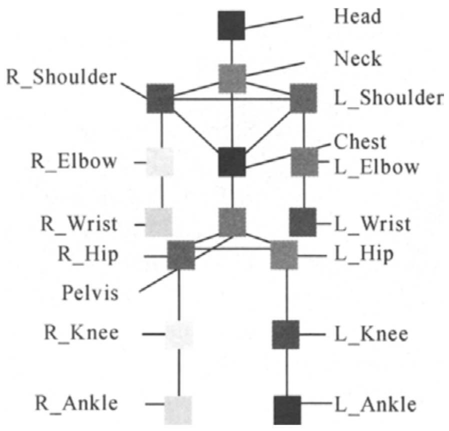


Fig. 3. 2 Human model and color markers on joints

It is very important to select a suitable color space in operations such as color edge detection and color histogram estimation. Most existing algorithms on color images can be viewed as extensions of the corresponding algorithms on intensity images. Consequently, many researches aim to exploit some information contained in the color images that is not present in intensity images [8]. Ohta, et al. [9] proposed a widely used method that can extract a group of orthogonal features. We select three fixed orthogonal features (equations (3-1)—(3-3)) in edge detection and histogram analysis. Experiments show that this color space is superior to many other ones.

$$I_1 = (R + G + B) / 3 \tag{3-1}$$

$$I_2 = (R - B) / 2 \tag{3-2}$$

$$I_3 = (2G - R - B) / 4 \tag{3-3}$$

3.1.2 Kalman Filter

We view the color markers' positions in the image sequence as a dynamic system. Use $\boldsymbol{P}$ to denote the position in an image. The position $\boldsymbol{P}=(x,y)^{\mathrm{T}}$ can be expressed as $\boldsymbol{P}=\boldsymbol{P}'+\boldsymbol{\eta}$, where $\boldsymbol{P}'$ is the actual coordinate. $\boldsymbol{\eta}$ is a 2D Gaussian random noise with mean value $\mathbf{0}$ and covariance $\boldsymbol{R}$. The trajectory of point $\boldsymbol{P}$ can be expressed as a 3rd polynomial. We define the state variable as $\boldsymbol{S} = (\boldsymbol{P}, \boldsymbol{P}', \boldsymbol{P}'')$, where $\boldsymbol{P}' = (x', y')^{\mathrm{T}}$ denotes $\boldsymbol{P}$'s velocity and $\boldsymbol{P}'' = (x'', y'')^{\mathrm{T}}$ denotes $\boldsymbol{P}$'s acceleration. Kalman filter is employed to predict the position in the next frame. Fig. 3. 3 shows Kalman filter's prediction result for the chest joint. It can be seen that the performance of Kalman filter is satisfactory.

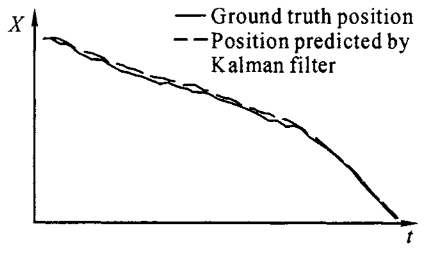


Fig. 3. 3 Prediction result of Kalman filter

3.1.3 Edge Detection and Edge Extraction

Here the goal of tracking is to get joints' positions for each frame, which implies two steps: color markers extraction and color markers' matching. In order to simplify following processes, we introduce the window concept which enables simultaneous color markers extraction and color markers' matching.

- In the first frame, the model's pose is constrained to stand idly. Thus, the rough positions of all joints can be easily estimated. Based on these first predicted positions, a window of size $M \times M$ can be defined for each joint and the subsequent tracking algorithms are constrained within the windows.
- In frame $n+1$, the windows are defined as centered at the position pre-

dicted by the Kalman filter. The size of the windows can be dynamically adapted to ensure that each window contains an entire color marker.

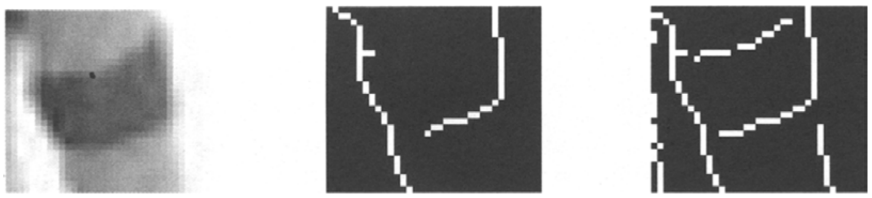
3.1.3.1 Edge Detection on Color Images using Canny Method

The conventional Canny edge detection method [10] contains three steps: (1) Filter the grayscale with 2D Gaussian filters to smooth the image; (2)Convolve the smoothed image with Canny operator to get gradients at each pixel; (3)Detect edges using the gradients by some rules.

The conventional Canny method can only be applied to grayscales. The frame images we have here are color images, so we apply Canny method to detect edges on three images composed of the three orthogonal components defined in Sect. 3. 1. 1 and apply an OR operation to get the final edges. Experiments show that this method can significantly reduce the lost edges. After the gradients are calculated, Canny method relies on two parameters T_1 and T_2 to control the judgment of edges. For each pixel, the rules are: (1) If the gradient is larger than T_1, it is an edge pixel; (2) If the gradient is less than T_2, it is not an edge pixel; (3) If the gradient is between T_2 and T_1, judge by the following sub-rule.

If a path starting at this current pixel and ending at some edge pixel can be found with the gradients on all pixels on the path being larger than T_2, label the current pixel and all pixels on the path as edge points.

In our case, edges are detected on an image sequence, and some interframe correlations will be present by setting dynamic T_1 and T_2 as threshold. Values of T_1 and T_2 are reduced at places where edges are present in the previous frame to increase the probability of edges being detected in the current frame, while values of T_1 and T_2 are increased at places where no edges are present in the previous frame to reduce noises. Fig. 3. 4 shows the comparison of the improved Canny method and the original method. It can be seen that the improved method detects more meaningful edges without introducing significant noisy edges.



markers in original frame conventional Canny method's result improved method's result

Fig. 3. 4 Comparison of conventional Canny method and the improved method

3.1.3.2 Edge Extraction Based on Hough Transform

After the edge detection, we get binary images indicating whether a pixel is an edge pixel or not. The edges in the binary images are not continuous and what we are concerned is whether they can form some known curves. Hough transform is an effective method to deal with this problem. Hough transform converts the image to a parameter space, where the parameters are represented by grids, and the best parameter grid is searched for. Hough transform has excellent tolerance and robustness in the case of discontinuous edge and noises and is very suitable in this scenario. There are two kinds of color markers used in this approach: circle (corresponding to the head) and rectangle (corresponding to markers on other joints).

Circle Extraction

Three parameters are needed to represent a circle in the Cartesian coordinate system, so the complexity of Hough transform is $O(k^3)$. In order to reduce the complexity, we adopt a two-step method.

In the first step, we try to detect the circle's center rather than the radius R. In the polar coordinate system, a circle can be represented as:

$$X_0 = X - R\cos\theta \tag{3-4}$$

$$Y_0 = Y - R\sin\theta \tag{3-5}$$

where (X_0, Y_0) is the center's coordinate and (X, Y) is the coordinate of pixels on the circle. The local gradient can be calculated by:

$$g = (g_x^2 + g_y^2)^{1/2} \tag{3-6}$$

Our improved method makes R vary in the normal direction of (X, Y) rather than all directions. Therefore, we can directly calculate the sine and cosine values in equations (3-4) and (3-5):

$$\cos\theta = g_x / g \tag{3-7}$$

$$\sin\theta = g_y / g \tag{3-8}$$

Combining the above equations, we can get:

$$Y_0 = X_0 \tan\theta + Y - X\tan\theta \tag{3-9}$$

The position of circle's center can be determined by equation (3-9). Here, only two parameters are still present and so the complexity is reduced to $O(k^2)$.

In the second step, we can get the radius R in the 1D parameter space:

$$R = \sqrt{(X - X_0)^2 + (Y - Y_0)^2} \tag{3-10}$$

Line Extraction

The extraction of line is simpler. In order to avoid unlimited slope, lines are represented using parameters in the normal direction:

$$X\cos\theta + Y\sin\theta = p \tag{3-11}$$

Voting mechanism is used to get the lines. Some inter-frame correlations will be present and the line information in the previous frame is exploited to improve the performance. Specifically, we make the grids near the value on the previous frame thinner and the grids far away from the values on the previous frame thicker in order to increase the accuracy without notably increasing the complexity. Fig. 3. 5 shows a line extraction result.

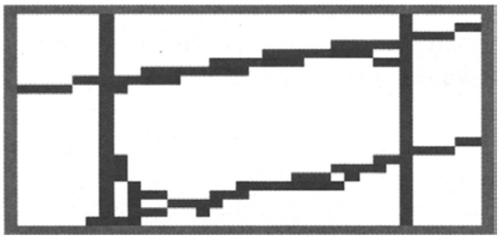


Fig. 3. 5 Line extraction result of Hough transform

3. 1. 4 Rectangle Construction

Due to the noises and the deformation of the color blocks, in most cases it is very difficult to extract four lines as the rectangle boundaries. After Hough transform, eight lines with highest probabilities are reserved and now our task is to select four lines from the eight to approximate a rectangle. In this section, two methods are proposed: Algorithm 1, which is based on the boundary analysis, and Algorithm 2, which exploits the information from the previous frame.

Algorithm 1: Rectangle Construction By Boundary Analysis

In Sect. 3. 1. 3, each extracted line E_i can be represented by two parameters: $E_i = \{\theta_j, p_j\}$ $(i=1,\cdots,8)$. On the other hand, the boundary lines of a rectangle has the relation: $E_i // E_j$ or $E_i \perp E_j$. Let E_0 denote the extracted line with the highest probability, then next we'd like to find the line that is parallel to E_0 and two lines that are perpendicular to E_0.

The parallel relation implies:

$$|\theta_i - \theta_0| < \varepsilon_\theta \cap |\theta_i + \theta_0 - 180| < \varepsilon_\theta \tag{3-12}$$

$$|p_i - p_0| > \varepsilon_p \tag{3-13}$$

The perpendicular relation implies:

$$||\theta_i - \theta_0| - 90| < \varepsilon_\theta \tag{3-14}$$

Equation (3-12) ensures that parallel lines have the same orientation and equation (3-13) ensures that two parallel lines are different. The line parallel to E_0 can be found using equations (3-12) and (3-13). One of the two lines perpendicular to E_0 (we call this line E_c) can be found using equation (3-14) and the other perpendicular line can be found using its parallel relation with E_c.

Because the reserved eight lines have different probabilities, in the above calculation we assign lines with higher probabilities with higher priorities. This helps to eliminate the effect caused by noises.

Sometimes due to the self-occlusion, one or more boundary lines are lost. In the case of more than one boundary lines being lost, the rectangle construction will fail (causing the system to resort to Algorithm 2 for the rectangle construction), while the case of one line being lost can be solved within this Algorithm 1 by automatically compensating the lost line. Given a line L_1 and a center predicted by Kalman filter (X_0, Y_0),

$$L_1 = Y\sin\theta + X\cos\theta \tag{3-15}$$

then the line at L_1's symmetric position with regard to (X_0, Y_0) (we call this line L_2) is:

$$L_2 = Y\sin\theta + X\cos\theta \tag{3-16}$$

Combining equation (3-15) and (3-16), we get:

$$L_2 = 2\times(Y_0\sin\theta + X_0\cos\theta) - L_1 \tag{3-17}$$

Algorithm 2: Rectangle Construction Based on the Previous Frame

Suppose the four boundary lines derived in the previous frame are $E_i' = \{\theta_i', p_i'\}(i=1,\cdots,4)$. In the current frame, we select $E_k'(k=1,\cdots,4)$ from the eight lines with the highest probabilities as the boundary lines in the current frame by the criterion of minimizing S:

$$S = F(k)\cdot(W_\theta\cdot|1/(\theta_k-\theta_k')| + W_p\cdot|1/(p_k-p_k')|) \tag{3-18}$$

The eight lines are sorted by descending probabilities, so $F(k)$ is monotonously descending. W_θ, W_p are two weights. Considering the angle difference is more effective, we set $W_\theta > W_p$. Now four boundary lines for the current frame can be selected using equation (3-18).

The usage of Algorithms 1 and 2 is as follows.

- Algorithm 1 only considers the state in the current frame, so it is not affected by the errors appearing in the previous frame. In addition, Algorithm 1 can detect the lost line and automatically synthesize a line for compensation. However, the drawback is that error lines might be introduced. Therefore, in Algorithm 1, the line extraction result must be verified using color and shape information. If the verification fails, then the system resorts to Algorithm 2 for alternative method.
- Algorithm 2 exploits the inter-frame information and in most cases it is robust. The drawback is that errors in the previous frame will be propagated to the current frame. Therefore, we use Algorithm 2 only as a complement to Algorithm 1.

3.1.5 Block Matching Algorithm

For the sake of robustness, in each frame, the derived rectangles in Sect. 3.1.4 have to be verified. In the case of severe self-occlusion, the verification may fail. In this section we present a new matching algorithm that searches for the best rectangle using the block's whole color information. This algorithm can deal with the tracking error in Sect. 3.1.4 due to self-occlusion. But this algorithm's efficiency is low and errors can propagate. So this algorithm is only used when the algorithms in Sect. 3.1.4 fail.

Liu, et al.[2] proposed a method that searches for a best matching block along a path that surrounds the position predicted by Kalman filter. However, the efficiency of the method is very low. Considering the inter-frame information, we assume that the color blocks in two successive frames only deform slightly. In this section, we propose a method with higher efficiency that takes two steps: matching by histogram and matching by morphing blocks.

Matching by Histogram

The bounding rectangles of the color marker in the previous frame are selected and the histogram is constructed. The histogram has some characteristics (see Fig. 3.6): (1) It has three peaks, with one major peak and two minor peaks; (2) The three peaks imply the marker, the clothes and the background, respectively.

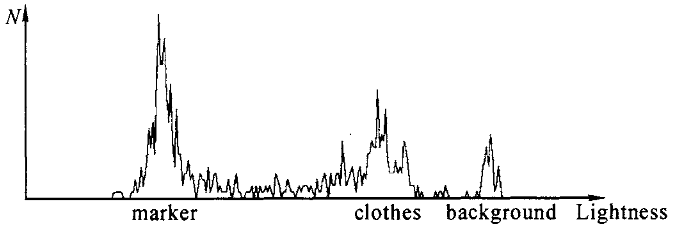


Fig. 3.6 Histogram of the bounding rectangle

Let (P_1, P_2, P_3) denote the three peaks on the histogram of frame N and let (P_1', P_2', P_3') denote the three peaks on the histogram of frame $N-1$, then the best block matching can be found by the following equations:

$$S_i = (\alpha \cdot |X(P_i) - X(P_i')| + \beta \cdot |Y(P_i) - Y(P_i')|) \qquad (3\text{-}19)$$

$$S = \sum_{i=1,2,3} W_i \cdot S_i \qquad (3\text{-}20)$$

where $X(P)$ means the X coordinate (i. e. color value) of point P, and $Y(P)$ means the Y coordinate of point P. α, β, W_i are weights. The value S in equation (3-20) represents the block matching degree between the two frames. In this way, we can get a rough position at which the histogram is most similar to that of the last frame.

Matching by Morphing Blocks

Based on the matching by histogram, now we use matching of morphing blocks to get the accurate shape of the color markers [1]. In this way, the matching complexity is reduced to M^2 (histogram calculation) $+ K$ (matching by morphing blocks).

3.2 3D Recovery of Human Motion Data

This section deals with recovering 3D human motion information from 2D tracking results. Researchers have proposed different approaches for 3D recovery. In [11], a constraint-based triangle method was used for recovering 3D data. O'Rourke and Badler [12] described a system that can analyze human motion in image sequences. Chen and Lee [13] proposed to exploit the global smoothing assumption to solve the ambiguity in recovering 3D data.

In this section, an expanding model is proposed. First, the 3D positions of joints are determined by expanding from a starting point. Then, 3D information is recovered. Compared to other algorithms, our method gets continuous and smooth starting point by optimization in the searching space and uses the assumption of global smoothness to effectively solve the ambiguity problem.

3.2.1 Two-step Calibration

In perspective projection, each point $P_w(X_w, Y_w, Z_w)$ in the world coordinate frame is transformed to a point $P(u, v)$ on the projection plane by equations (3-21) and (3-22):

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \boldsymbol{R} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} - \boldsymbol{T} \tag{3-21}$$

$$(u, v) = \left(\frac{f \cdot X_c}{Z_c}, \frac{f \cdot Y_c}{Z_c} \right) \tag{3-22}$$

where $\boldsymbol{R}$ is a rotation matrix, $\boldsymbol{T}$ is a translation matrix and f is the focal length. The unknown variables, $\boldsymbol{R}$, $\boldsymbol{T}$ and f, can be derived given some 3D points and the corresponding 2D points. When the tracking result is available, the 2D projections of features are known and the 3D position in the world coordinate frame can be arbitrarily designated by the system. As long as the proportions between the features are consistent to the reality, we can always find the location and orientation of camera in the world coordinate frame (i. e. determine $\boldsymbol{R}$, $\boldsymbol{T}$ and f) which can make the system-designated 3D points and the 2D positions on the image converge. Below is the detailed procedure.

Step 1 The first step is to solve a linear equation combination using least square method and find external parameters. Since $\boldsymbol{R}(r_1, r_2, \cdots, r_9)$, $\boldsymbol{T}(T_x, T_y, T_z)$ and f have 12 DOFs, which is larger than the number of point pairs that can be provided, intermediate variables have to be introduced: $T_y^{-1}r_i$, $T_y^{-1}T_x$, $i \in [1, 6]$. From equations (3-21) and (3-22) it can be shown that the intermediate variables must satisfy the following equation:

$$[Y_{di}x_{wi}, Y_{di}y_{wi}, Y_{di}z_{wi}, Y_{di}, -X_{di}x_{wi}, -X_{di}y_{wi}, -X_{di}z_{wi}] \cdot [T_y^{-1}r_1, T_y^{-1}r_2, T_y^{-1}r_3, T_y^{-1}T_x, T_y^{-1}r_4, T_y^{-1}r_5, T_y^{-1}r_6]^{\mathrm{T}} = X_{di} \tag{3-23}$$

where (x_{wi}, y_{wi}, z_{wi}) is the world coordinate of point i and (X_{di}, Y_{di}) is the corresponding point on the projection plane. If we have more than 7 point pairs, then the 7 intermediate variables can be determined by solving this equation combination. After that, we can determine $(r_1, r_2, \cdots, r_9, T_x, T_y)$ by using the orthogonal property of $\boldsymbol{R}$. If we constrain the points to lie on the same plane and let $z_{wi}=0$, then the 3rd and 7th components of the above equation are zero and a minimum of 5 point pairs is sufficient to solve the equation combination. Besides, in the case of point pairs lying on the same plane, matrix $\boldsymbol{R}$ is more suitable for orthogonality. In practice, the common plane method is adopted in our system.

Step 2 T_z and f can be calculated by the following equation:

$$(y_i, -d_y Y_i)\begin{pmatrix} f \\ T_z \end{pmatrix} = w_i d_y Y_i \tag{3-24}$$

where d_y is the physical size of each pixel along y direction,

$$\begin{cases} y_i = r_4 \cdot x_{wi} + r_5 \cdot y_{wi} + r_6 \cdot z_{wi} + T_y \\ w_i = r_7 \cdot x_{wi} + r_8 \cdot y_{wi} + r_9 \cdot z_{wi} \end{cases}$$

In practice, we select the chest, abdomen, left shoulder, right shoulder, left hip and right hip as the point pairs. (X_{di}, Y_{di}) of each point is derived from tracking, and corresponding (x_{wi}, y_{wi}, z_{wi}) can be designated to any world coordinate as long as the proportions between the points are consistent to the reality. Because the camera is fixed during the clip, the camera calibration has to be conducted only in the first frame and the calibration is used for the entire clip. Therefore, we can constrain the subject to be in the idle pose at the beginning of the clip and make the six points on the same plane. Our method only requires solving two small-scaled linear equation combinations for all parameters $\boldsymbol{R}$, $\boldsymbol{T}$ and f. Compared with other non-linear calibration methods, our method has the advantages of high efficiency and robustness.

3.2.2 Selection of Start Points

After camera calibration, the next issue is how to get the 3D coordinates of joints from the constraint of perspective projection.

Since the triangle composed of chest, left shoulder and right shoulder best satisfies the non-deformation property of rigid movements and they are rarely occluded, we use this triangle as the starting points. Fig. 3.7 demonstrates the motion information associated with this triangle (for better comparison, the data are translated and scaled). In the real capture, the subject approaches the camera with constant speed and the Z coordinate should be an increasing and smooth curve. The figure also shows the Z coordinate determined by solving the linear equation combination. It can be seen that the results are not stable and cannot reflect the real human motion. Considering that the starting triangle represents the global motion trends and does not abruptly change, we want to get useful information from the smoothed and filtered 2D tracking results and use this information to derive the Z coordinate in the 3D space. The star curve in Fig. 3.7 shows the total length of the starting triangle and the circle curve shows the speed centroid of the starting triangle along the Z direction. For most motions, the variation of the depth information of the starting triangle is highly related to the total length of the triangle on the projection plane. When the subject is approaching the camera, the human's projection gets larger and the total length of the starting triangle increases, and vice versa. The star line in Fig. 3.7 shows this trend. Besides, we also consider the speed of the starting triangle's centroid (the bottommost line in Fig. 3.7). Based on the above analysis, we can divide the entire clip into segments, making the variation of depth of the starting triangle continuous in each segment. And then we can define a searching space and calculate the unknown parameter Z_i using the continuity property of motions and the non-deformation property of rigidity. The procedure is detailed in the following.

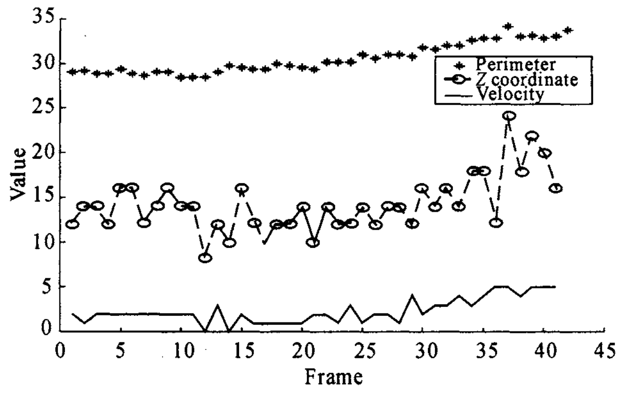


Fig. 3.7 The motion curves

Determining the Searching Space

Let the Z coordinates of the starting triangles of the nth frame be (Z_1', Z_2', Z_3') and the speed of Z coordinate be (V_1, V_2, V_3), then we can define the searching space for the $(n+1)$th frame to be $(Z_1' - \alpha V_1, Z_1' + \beta V_1)$. Because the variation of the starting triangle is gradual during the clip, we can constrain the coordinates of Z_2 and Z_3 using:

$$Z_2 \in (\theta \cdot Z_1 \cdot K_{21}, \sigma \cdot Z_1 \cdot K_{21}) \qquad (3\text{-}25)$$

$$Z_3 \in (\theta \cdot Z_1 \cdot K_{31}, \sigma \cdot Z_1 \cdot K_{31}) \qquad (3\text{-}26)$$

where $K_{21} = Z_2'/Z_1'$, $K_{31} = Z_3'/Z_1'$.

Determining the Optimization Criterion

Above all, it is an important criterion that the length of each edge of the starting triangle should be constant (S_L). Then, the motion's continuity is also considered, quantified by the calculation of acceleration (S_V). S_P stands for the difference between the normal vectors of the two frames. Finally, the optimal location can be derived by $\phi > \varphi > \gamma$:

$$S = \phi \cdot S_L + \varphi \cdot S_V + \gamma \cdot S_P \qquad (3\text{-}27)$$

Fig. 3. 8 shows the Z coordinate derived by our method. The experiment shows that the result of our method is significantly better than the method of solving non-linear equation combination. But, in our method, several parameters α, β, θ, σ are involved and their values can have a significant impact on the result. Therefore, we provide a user-friend interface for the user to interactively set the parameters. For example, for the walking motion, the motion along Z coordinate can be described by straight line with constant speed. Therefore, α is set to zero and β is set to a value larger than 1. For jumping motion, the normal direction of the starting triangle has a constant orientation and so the corresponding θ and σ are set to values near to 1.

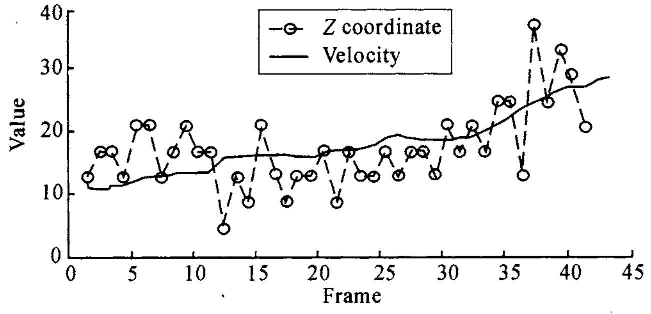


Fig. 3. 8 The motion curves obtained by our method

3.2.3 Solving for Other Joints

We've got the 3D coordinate of the starting triangle. And now the task is to expand from the starting triangle and get the 3D coordinates of other joints. It is known that in the perspective projection one point on the projection plane corresponds to a line in the 3D coordinate frame. In order to determine the real 3D position on the line, prior knowledge about the skeleton length or human model must be exploited. From an adjacent point whose 3D coordinate is known, we can search the line for a point that can make the distance between the two points in the 3D space the same as the real length on the human skeleton model. However, it can be seen in Fig. 3.9 that actually two points P_1 and P_2 both satisfy this criterion. This ambiguity is caused by the illness of the problem of recovering 3D information from 2D data and has to be eliminated by prior knowledge.

The method presented above first gets the set of all positions and finally uses an object function based on global motion smoothing assumption to select a single pose. This assumption ignores the motion's complexity and cannot guarantee to solve the ambiguity problem. Therefore, we try to analyze the features of the 2D motion. We find the frames with abrupt changes and divide the entire clip into segments so that the smoothing assumption can be better satisfied within each segment. Fig. 3.10 shows a clip of jumping motion with 33 frames. Using simple filter techniques, six zero-crossings can be identified (the 7th, 9th, 14th, 17th, 24th and 26th frames. By observing the original video clip, it can be discovered that these zero-crossings are highly consistent to the real abrupt change points in the motion. Based on these zero-crossings, the entire clip can be divided into

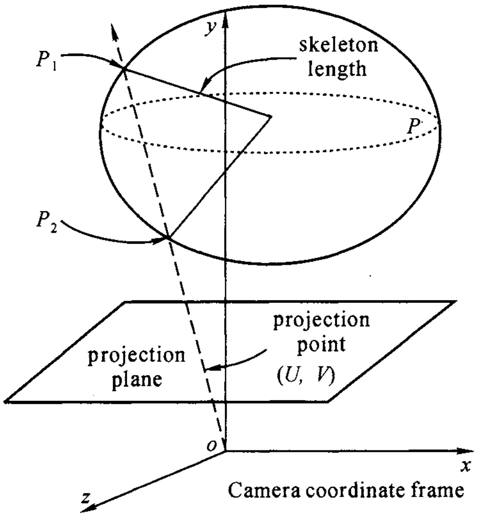


Fig. 3.9 The expanding method

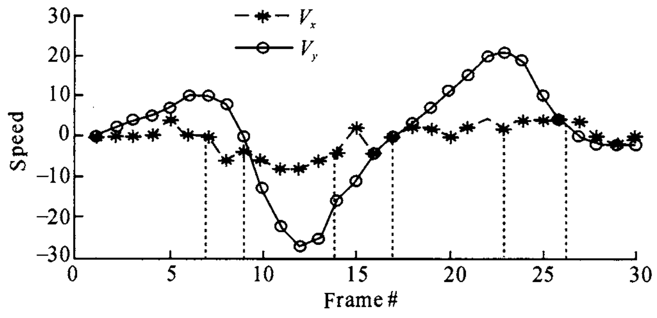


Fig. 3. 10 Hand motion curves of 2D image

independent segments. For each sub-segment (M_f, N_f) without abrupt changes, the ambiguity problem can be solved by the following procedure.

Step 1 For each joints, two candidate positions satisfy the perspective projection. Since there are three points in the starting triangle, the final data structure derived should be $3\times(N-M)$ independent expanding trees.

Step 2 Some unreasonable branches are pruned using prior knowledge. For example, in the camera coordinate frame, thighs, knees and ankles satisfy a constraint that the knee is always in front of the other two. In this way, some obviously unreasonable branches can be pruned. Besides, the interactive interface allows the user to define constraints. For some motions with strong regularities (e. g. walking, jumping, etc), typically a lot of constraints can be exploited.

Step 3 For those joints whose ambiguity cannot be solved by constraints, we use the motion continuity with the segments to solve the ambiguity. Now we raise a simplified example considering the expanding tree starting from the left shoulder which consists of left elbow and left wrist as two child joints and only one DOF is considered. We define a smoothing criterion function S for frames $i-1$, i and $i+1$:

$$S(i-1,i,i+1)=\bar{\omega}_{\text{elbow}}(i-1,i,i+1)+\bar{\omega}_{\text{wrist}}(i-1,i,i+1) \quad (3\text{-}28)$$

where $\bar{\omega}$ is the angular acceleration within the three frames. Based on the above equation, we can define:

$$\Phi(1,2,\cdots,N)=S(1,2,3)+S(2,3,4)+\cdots+S(N-2,N-1,N) \quad (3\text{-}29)$$

Φ measures the smoothing degree within N frames. For each frame, we introduce a transfer graph (Fig. 3. 11 is such a transfer graph when the segment length is 5) consisting of a starting node S and end node E. Thus, the optimization of searching for best trajectory is converted to the shortest path problem and the computation complexity is greatly reduced.

Recovering 3D motion from 2D data is a challenging problem. Our method exploits the global motion smoothing assumption to get a best trajectory for human body joints. Compared to methods considering only a single frame or two successive frames, our method is more suitable to the intrinsic continuity and smoothness properties of human motion. Besides, we find abrupt change points in the 2D data and divide the entire clip into segments. This strategy makes the smoothing assumption more realistic and significantly reduces the computational complexity. There are also some limitations of our method. First, the pruning of unreasonable branches in Step 2 typically relies on the prior knowledge, and for complex motions, reasonable branches might be pruned. Second, if a joint abruptly changes in two successive frames and the algorithm misjudges, the algorithm based on global consideration can make this misjudgment lead to errors for the entire decision.

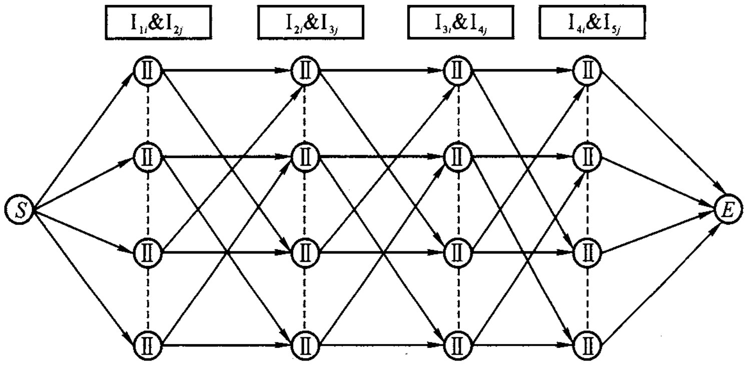


Fig. 3. 11 Transfer graph

3.3 Case Studies: VBHAS V2.0

We implement a system called Video-based Human Animation System (VBHAS V2. 0) based on the algorithm introduced above. The input of the system is a video clip taken in the lab containing a person wearing tight clothing and the output is the 3D coordinate sequence of each joint.

3.3.1 Results of Human Motion Tracking

We captured a video of walking at a constant speed (containing 45 frames) for experiment. Fig. 3. 12 shows the tracking results for the 1st, 13rd, 20th, 27th, 32nd and 42nd frames from left to right. The rectangles and dots show the tracked color markers. It can be seen that in the case with-

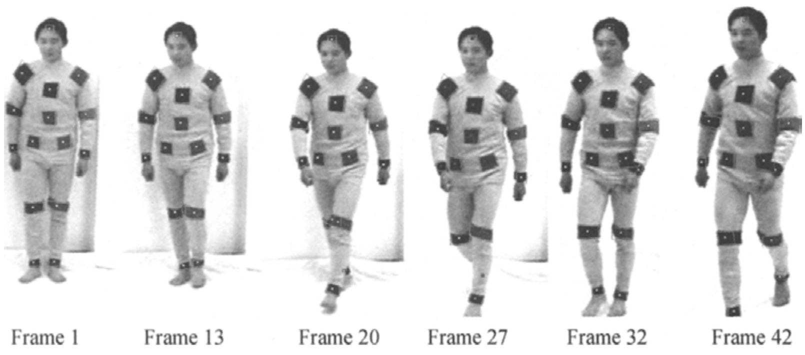


Fig. 3. 12 The tracking result of human motion

out self-occlusion, our algorithm can always get the correct position. In the case of partial occlusion (e. g. the knees in the 20th, 27th frames), the block matching algorithm introduced into our method can still get the correct tracking. In the 27th frame, the right ankle is completely occluded and the tracking of this joint is lost in that frame. However, in the subsequent frames, as the occlusion fades off, the system regains the correct tracking of that joint, indicating that the position in the 27th frame can be calculated by interpolation.

3. 3. 2 Results of Human Motion 3D Reconstruction

After joint tracking, VBHAS V2. 0 can recover the 3D human motion. The recovered 3D data are then sent to Poser 4 for animation. Fig. 3. 13 shows the result, in which the left columns are the captured video frames and the right columns are the animated motion. It can be seen that the system can capture the 3D human motion and retarget it to an animated model. However, only 14 color markers are used on the tight clothing, so some subtleties in the motion (e. g. motion of the hands and feet) will be lost. And this is one of our future works.

We summarize two characteristics of our tracking and reconstruction method. First, the results are very robust and accurate. Second, the incorporation of Kalman filter and block matching algorithms enables this system to deal with self-occlusion partially. Although this method poses some limitations on the model's clothing and capturing background, it is very useful in applications such as indoor motion analysis and animation producing. Besides, this method approaches different joints independently and stands a chance of parallelism.

Our approach which uses only one camera has the advantages of low cost and simple hardware over other methods using multiple cameras. In addition, it has the potential of exploiting the abundance of existing video clips in movies or other media. The limitation is its dependence on the prior knowledge, which prohibits VBHAS V2. 0 from reconstructing unlimit-

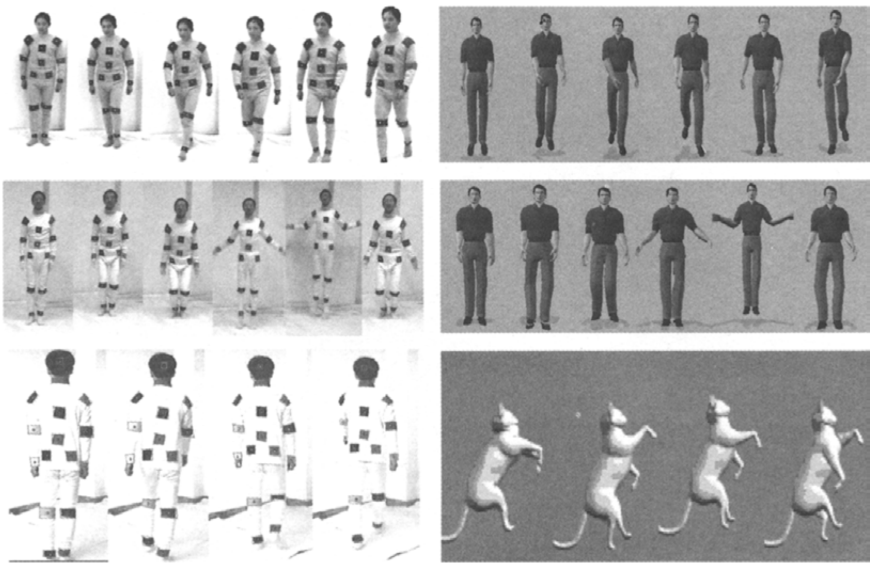


Fig. 3. 13 3D reconstruction and retargeting of the motion data

ed human motion, but does not affect its use in applications where prior knowledge of the motion is accessible.

References

1. Rehg J, Kanade T. Visual tracking of self-occluding articulated objects. Technical Report CMU-CS-TR-94, Carnegie Mellon University School of Computer Science, 1994.
2. Hogg D. A program to see a walking person. Image Vision Computing, 1(1): 5-20, 1983.
3. Segen J, Pingali S. A camera-based system for tracking people in real time. In: Proceedings of International Conference on Pattern Recognition, pp. 63-67, IEEE Computer Society, 1996.
4. Bregler C, Malik J. Video Motion Capture. In: SIGGRAPH'98, pp. 353-360, ACM Press, 1998.
5. Hager GD, Belhumeur PN. Real time tracking of image regions with changes in geometry and illumination. In: CVPR'98, pp. 403-410, IEEE Computer Society, 1998.
6. Papanikolopoulos N, Khosla P, Kanade T. Visual tracking of a moving target by a camera mounted on a robot: a combination of control and vision. IEEE Transactions on Robot Automation, 9(1): 13- 35, 1993.
7. Kalman RE. A new approach to linear filtering and prediction prob-

lems. Transactions of the ASME -Journal of Basic Engineering, 82: 35-45, 1960.

8. Shafer S. Using color to separate reflection components. Color Research and Application, 1985, 10(4): 210-218, 1985.
9. Ohta Y, Kanade T, Sakai T. Color information for region segmentation. Computer Graphics and Image Processing, 13(3): 222-241, 1980.
10. Canny JF. A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6): 679-698, 1986.
11. Barnard ST, Fischler MA. Computational stereo. Computing Surveys, 14(4): 553-572, 1982.
12. Steketee SN, Badler NI. Parametric keyframe interpolation incorporating kinetic adjustment and phrasing control. Computer Graphics, 19(3): 255-262, 1985.
13. Chen Z, Lee HJ. Knowledge-guided visual perception of 3D human gait from a single image sequence. IEEE Transactions on Systems, Man, and Cybernetics, 22(2): 336-342, 1992.

4

Two-camera-based Human Motion Capture

In order to extract complex human motion precisely, multiple cameras are often used to capture the video sequences, then tracking and reconstruction of human motion can be achieved by virtue of the multi-view video sequences[1]. The self-occlusion problem that occurred during tracking can also be solved with the multi-view pattern. Multiple views mean that the same scene is captured with the same sampling rate from different viewpoints. According to the principle of vision, multiple corresponding image feature points are competent for reconstructing 3D coordinates of feature points accurately. Therefore, compared with monocular video sequence, 3D reconstruction is easier under multi-viewpoints. However, difficulties of feature correspondence and self-occlusion also exist in the tracking of multi-view video sequences. Especially, the automatic corresponding of multiple views is still a challenging issue. In this chapter, we intend to give readers more insight into the two-camera-based human motion capture as well as the VBHAS V3. 0 (Video-based Human Animation).

VBHAS V3. 0 uses two cameras to capture the video sequence. First, we introduce the image pre-processing and image synchronization. Then, the automatic tracking of human features is discussed in Sect. 4. 2. In Sect. 4. 3, we discuss the 3D motion reconstruction. And Sect. 4. 4 will give a demonstration of VBHAS V3. 0.

4. 1 Human Model

Frequently used human models include stick model, silhouette model and volume model. Though the motion feature extraction is easier when silhouette model and volume model are used, the computational complexity of feature correspondence is higher in these cases; on the other hand, if stick model is used, the feature extraction is more difficult. Therefore, we need to balance between model complexity and the computational complexity of feature correspondence. In VBHAS V3.0, we propose to regard human body as a set of rigid bodies connected by articulated joints and simplify the human motion as the skeleton motion. As shown in Fig. 4.1, the human model is composed of 16 joints as well as the rigid bodies connecting the joints[2]. To capture the motion information, we develop a novel kind of maillot (Fig. 4.2) according to the proposed human model. Each joint is decorated with color block in a specific kind of color. Usually, the color block is independent of each other, therefore we treat each color block as a single feature, and the human motion can be obtained by tracking the color blocks and building the correspondence between features in different video frames.

In two-camera-based human motion tracking, multiple techniques are adopted, including Kalman filter, epipolar line, attribute quantification, incomplete motion feature tracking and HMM based tracking. In the following sections, we will discuss these approaches in detail.

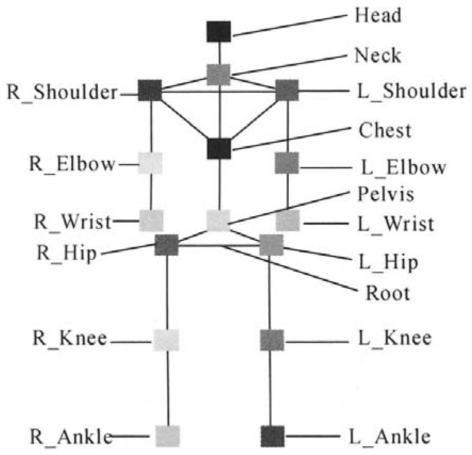


Fig. 4.1 Color block based human motion model

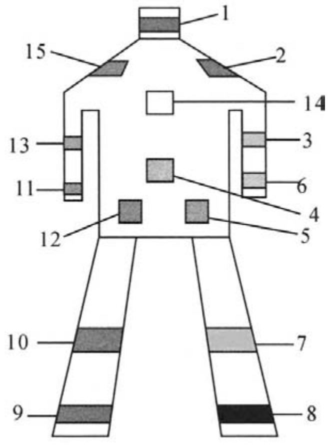


Fig. 4.2 Color block based maillot

4.2 Human Motion Feature Tracking

4.2.1 Feature Tracking Algorithms Based on Kalman Filter and Epipolar Constraint

4.2.1.1 The Principle

Only using single image feature (color, texture, shape) is incompetent for motion tracking when salient motion or self-occlusion occurs, or high precision is needed. Other constraints should be incorporated to improve the efficiency and accuracy of motion tracking. According to video analysis, it is found that: (1) The movement of feature points in one frame is correlated to that in the adjacent frame, so Kalman filter contributes to reducing the search space and improving the efficiency and accuracy of motion tracking; (2) In motion capture, the epipolar line obtained by multi-camera calibration can be used to guide the feature tracking, thus reducing the error that occurs during tracking.

Experiments show that:

- Considering the attributes of feature points and its neighbors, the sub-block based tracking performs better than the point-based matching;
- Though the tracking algorithm executed in RGB space is adaptive to frames with variant lighting conditions, the error is evident to some joints with salient motion;
- Gray scale value based tracking is more sensitive to lighting condition, therefore, the tracking results are better when the difference in gray scale value between background scene and moving objects is large;
- Kalman filter can improve the tracking result of motion feature evidently, however, if there exists serious error in previous frame, the tracking result of current frame degenerates greatly. This problem can be solved by setting a threshold for Kalman filter.

Based on the discussion above, we propose a novel feature tracking algorithm. The basic workflow is: first, predict the position of feature point in next frame using Kalman filter; then, search for the seed point using sub-block based matching in RGB and gray scale space, and cluster the color blocks centering around the seed point; last, the center of color blocks is the tracking result. After obtaining the feature points in one frame, use the epipolar line to guide the tracking in the video captured by the other camera simultaneously.

4.2.1.2 Tracking Algorithm

According to the principle we just described, we develop our tracking algorithm as follows ($C_1(i)$ denotes the ith frame captured by Camera 1, and $C_2(i)$ denotes the ith frame captured by Camera 2).

Step 1 Find the mark points in both videos, and synchronize the videos by use of the mark points.

Step 2 To synchronized videos captured by two cameras, mark the joint feature points of human model manually on the first frame, initialize the speed and acceleration of 3D feature points as 0 (in order to deal with the self-occlusion), and set the initial speed of 2D feature points to be 0 also.

Step 3 Predict the position and speed of 2D feature points in the $(i+1)$th frame using Kalman filter in the image coordinate system.

Step 4 Track the feature points in frame $C_1(i+1)$ according to the prediction as follows.

(1) Search the matching points corresponding to the feature points in $C_1(i)$ based on sub-block matching. The eight-neighbor search space centers around the predicted feature points in frame $C_1(i+1)$, and the matching is performed in RGB and gray scale space respectively.

(2) Let the matching point be the seed point, and cluster all the points in a bounding box (determined by the focal length) according to a specific threshold (determined by the average gray scale value of sub-block).

(3) Solve for the centers of all the clusters, and the center points are actually the feature points in frame $C_1(i+1)$.

(4) Modify the state vectors according to the tracking results (2D and 3D state vectors).

Step 5 Guide the tracking of the corresponding feature points in frame $C_2(i+1)$ based on the epipolar constraint equation.

(1) Track the feature points in frame $C_2(i+1)$ using the method in Step 4, and solve for the candidate points.

(2) If the distance between a candidate feature point and the epipolar line is larger than a specific threshold, remove this feature point from the candidate feature point set.

(3) Search for the matching point corresponding to the seed point according to the search route determined by epipolar line. Let the matching points be the candidate feature points.

(4) Compute the distance between candidate feature points and the epipolar line, and select the feature point with the shortest distance as the tracking result.

(5) If no points satisfying the conditions are found (due to occlusion), go to (6);

(6) Modify the state vectors, go to (3).

Step 6 If the search around the predicted points fails (due to self-occlusion), use Kalman filter under world coordinate system. In this case, we regard the 3D points as the tracking results, and regard the predicted image points as the 2D tracking results. When the self-occlusion disappears, modify the state vectors using the actual tracking results.

Step 7 If tracking for next frame is needed, go to Step 3; else, the algorithm terminates.

4.2.1.3 Search Route and Matching Criterion

Same route is adopted when searching for the seed points and clustering the color blocks. The searching route starts from the predicted center point, and goes through the eight-neighbor space anticlockwise, see Fig. 4. 3.

Due to the relationship between gray scale value and environment condition, when it is dark, the gray scale value of foreground is similar to that of background. Therefore, we adopt RGB and gray scale value for matching. When clustering the color block, we only use the gray scale value.

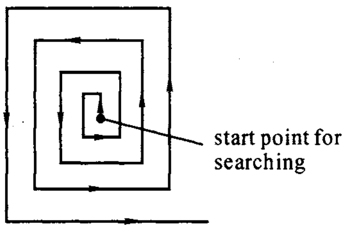


Fig. 4. 3 The search route

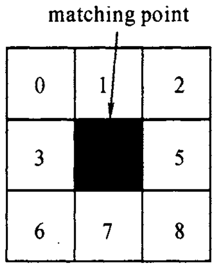


Fig. 4. 4 The sub-block

Search for the Seed Points

When searching for the seed points corresponding to the feature points in the previous frame, we adopt a sub-block based method. The sub-block comprises of the feature point to be matched and its eight-neighbors, see Fig. 4. 4. First, to a specific point p, we compute the sum of the square of the difference between p related sub-block and corresponding sub-block in the previous frame, in R, G, and B channels independently:

$$RedDiff = \sum_{i=0}^{2}\sum_{j=0}^{2}(p[i][j].\mathrm{red} - Feature_m[i][j].\mathrm{red})^2$$

$$GreenDiff = \sum_{i=0}^{2}\sum_{j=0}^{2}(p[i][j].\mathrm{green} - Feature_m[i][j].\mathrm{green})^2$$

$$BlueDiff = \sum_{i=0}^{2}\sum_{j=0}^{2}(p[i][j].\mathrm{blue} - Feature_m[i][j].\mathrm{blue})^2$$

where $p[i][j]$ $(i,j=0,1,2)$ denotes each pixel in the p related sub-block, $Feature_m[i][j]$ $(i,j=0,1,2)$ denotes each pixel in the sub-block related to the mth feature point in the previous frame.

The matching criterion can be described as following two formulas:

$$\min\ (RedDiff \cap GreenDiff \cap BlueDiff) \tag{4-1}$$

$$\| p.\mathrm{gray} - MeanGray_m \| \leqslant Deviation \tag{4-2}$$

Equation (4-1) means no point exists which has smaller R, G, and B value. This is because in many cases, the R, G and B value of the point differs greatly from that of the feature point, while the sum of square of the difference in R, G and B value is smaller. The matching point obtained in this way may be "false point" due to noises. Equation (4-2) indicates that the deviation between gray scale value of the matching point and the average value is below some threshold, which ensures the matching point is in the sub-block. When both equation (4-1) and (4-2) are satisfied, the point is the cluster center point.

Cluster the Color Block Region

The aim is to find the color block region, and the tracking result is the center of the color block region. In the previous section, we have found the start point of clustering. Because the gray scale value reflects the statistical characteristics of a point's R, G, B value, we only adopt gray scale value to cluster the color block region. The criterion for clustering color block is the same as equation (4-2).

4.2.1.4 Kalman Filter for Feature Point Prediction

In feature point tracking, the marks attached on the joints may be occluded by human motion, consequently accurate feature point tracking is not available. The problem can be solved by use of Kalman filter to predict the feature points. Under ideal condition, Kalman filter under world coordinate system reflects the actual motion of feature points, however, in practice, considerable errors may occur when projecting the 3D feature points to 2D image plane because of the uncertainty in calibration. So, in our VB-HAS, Kalman filter under image coordinate system is used for predicting the feature point when no self-occlusion occurs, while if there exists self-occlusion, Kalman filter under world coordinate system is adopted.

Kalman Filter under World Coordinate System

Human motion can be regarded as a dynamic system. The motion style of joints in adjacent frames can be regarded as evenly accelerating. Cubic polynomial can be used to describe the motion trajectory of point p along x (y or z) direction. Kalman filter is adopted for prediction in x, y or z direction. Suppose $\boldsymbol{S}=(P_i, V_i, a_i)^{\mathrm{T}}$ is a state vector where P_i is the coordinate offset of p along x, y or z direction, V_i is the speed offset of p along x, y or z direction, and a_i denotes the acceleration offset of p along x, y or z direction. The state equation can be described as:

$$\boldsymbol{S}(k+1) = \boldsymbol{F} \cdot \boldsymbol{S}(k) + \boldsymbol{G} \cdot \boldsymbol{n}(k) \tag{4-3}$$

where

$$\boldsymbol{F}=\begin{bmatrix} 1 & T & \frac{1}{2}T^2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{G}=\begin{bmatrix} \frac{1}{2}T^2 \\ T \\ 1 \end{bmatrix},$$

and $k=0,1,2,\cdots$ means the number of each frame, $T=t_{k+1}-t_k$ is the time interval, $\boldsymbol{n}(k)$ describes the noise of acceleration along x, y or z direction.

Suppose $\boldsymbol{n}(k)$ is Gaussian, with zero mean and covariance matrix $\boldsymbol{Q}$.

The feature point $\boldsymbol{P}=(x,y,z)^{\mathrm{T}}$ can be described as $\boldsymbol{P}=\boldsymbol{P}'+\boldsymbol{\eta}$ where $\boldsymbol{P}'$ is the noiseless observation of joint feature point, $\boldsymbol{P}$ is the actual observation, $\boldsymbol{\eta}$ is Gaussian with zero mean and covariance matrix $\boldsymbol{R}$. Therefore, the observation equation is:

$$\boldsymbol{X}(k) = \boldsymbol{H} \cdot \boldsymbol{S}(k) + \boldsymbol{\eta}(k) \tag{4-4}$$

where $\boldsymbol{H}=[1,0,0]$ is a 1×3 matrix.

Kalman Filter under Image Coordinate System

In feature tracking algorithm, if we project the point predicted by Kalman filter under world coordinate system to image plane, and select the projection as the seed point for tracking, the experimental result is not good. In terms of image feature point tracking, we use Kalman filter under image coordinate system for prediction. Due to the tiny time interval between adjacent frames, we suppose the corresponding feature points in adjacent frames move at even speed. However, the moving speed of feature points in arbitrary two frames is different and the speed can be constantly updated according to the tracking results. The state vector of the system can be described as $[u(k), v(k), \dot{u}(k), \dot{v}(k)]^{\mathrm{T}}$ where $u(k)$ and $v(k)$ denote the coordinate offset along u and v direction in image coordinate system respectively, while $\dot{u}(k)$ and $\dot{v}(k)$ denote the speed offset of feature point along u and v direction. Thus, the state equation of the dynamic system is as follows:

$$\boldsymbol{X}(k+1) = \boldsymbol{F} \cdot \boldsymbol{X}(k) + \boldsymbol{\eta}(k) \tag{4-5}$$

where $\boldsymbol{\eta}(k)$ is Gaussian white noise with zero mean and covariance matrix $\boldsymbol{Q}$.

The observation equation listed below describes the relationship of noiseless tracking results and practical ones.

$$\boldsymbol{Z}(k)=\boldsymbol{H} \cdot \boldsymbol{X}(k) + \boldsymbol{W}(k) \tag{4-6}$$

where

$$\boldsymbol{F}=\begin{bmatrix}1 & 0 & T & 0\\ 0 & 1 & 0 & T\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix},\quad \boldsymbol{H}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\end{bmatrix},$$

and T is the time interval between video frames, $\boldsymbol{W}(k)$ is Gaussian white noise with zero mean and covariance matrix $\boldsymbol{R}$.

State prediction and modification of dynamic system is similar to Kalman filter of 3D feature points.

4.2.1.5 Epipolar Constraint Guided Feature Tracking and Artifact Elimination

In multi-camera tracking, the constraint between multiple views can be used to guide the feature tracking and eliminate the artifact of tracking results.

In camera calibration, the projection matrices of two cameras can be described as follows:

$$Z_{C_1}\boldsymbol{u}_1 = \boldsymbol{M}_1\boldsymbol{X}_p = (\boldsymbol{M}_{11}, \boldsymbol{m}_1)\boldsymbol{X}_p \tag{4-7}$$

$$Z_{C_2}\boldsymbol{u}_2 = \boldsymbol{M}_2\boldsymbol{X}_p = (\boldsymbol{M}_{21}, \boldsymbol{m}_2)\boldsymbol{X}_p \tag{4-8}$$

where Z_{C_1} and Z_{C_2} are the z coordinates of point p in the camera coordinate system, $\boldsymbol{X}_p$ is the homogeneous coordinate of p in the world coordinate system, $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are the homogeneous coordinates of p's projections p_1 and p_2 on the image plane, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are the transformation matrices, $\boldsymbol{M}_{11}$ and $\boldsymbol{M}_{21}$ are the left side 3×3 matrices of $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$, $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ are the right side 3×1 vectors. Equations (4-7) and (4-8) thus can be converted to:

$$\boldsymbol{m} = \boldsymbol{m}_2 - \boldsymbol{M}_{21}\boldsymbol{M}_{11}^{-1}\boldsymbol{m}_1 \tag{4-9}$$

$$\boldsymbol{u}_2^{T}[\boldsymbol{m}]_x\boldsymbol{M}_{21}\boldsymbol{M}_{11}^{-1}\boldsymbol{u}_1 = \boldsymbol{0} \tag{4-10}$$

Equations (4-9) and (4-10) denote the epipolar constraints that $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ should satisfy. When $\boldsymbol{u}_1$ is given, the equations are epipolar constraints related to $\boldsymbol{u}_2$ in the other image coordinate system.

Based on the constraints, we use the tracking results in $C_1(i)$ to guide the tracking in $C_2(i)$, and the search route is shown in Fig. 4. 5. Epipolar constraints are used to judge the accuracy of tracking in $C_2(i)$, the criterion is whether the distance between the tracked point and the epipolar line surpasses a certain threshold.

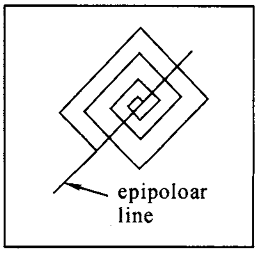


Fig. 4. 5 Searching route guided by epipolar constraints

4. 2. 2 Feature Tracking Based on Attribute Quantification

4.2.2.1 The Principle

In video-based motion capture, due to the salient human motion and self-occlusion of human movement, the variation of pixel brightness is very large, thus the pixel value based tracking algorithm lacks robustness [3]. Though the information of single pixel is limited, the information of each image feature remains abundant, such as position, size and mass. In video sequence of high sampling rate, the position and size of features in adjacent frames are very similar, and the speed and acceleration of feature points are predictable. To the isolated feature, the brightness, size, position and speed can be used for feature correspondence in motion tracking; while to the features whose motion trajectory is discontinuous, the speed and position can be used to guide the feature correspondence. Therefore, when multiple features exist in the scene, adaptively integrating the feature attributes can improve the robustness of human motion tracking.

According to the above discussion, we propose a feature tracking algorithm based on feature attribute quantification. The workflow is: first, find the human features in image, extract the features as well as the feature attributes and compute the matching factor between corresponding features in adjacent frames; second, build the matching matrix for weighted sum of each feature attribute according to the importance of each attribute during tracking; last, search for the optimized feature correspondence in matching matrix by greedy algorithm, thus construct the correspondence for the features in adjacent frames.

4.2.2.2 Tracking Algorithm

Step 1 Suppose self-occlusion does not occur in the first frame, track the feature joints in the first frame by model matching.

Step 2 Track the frame $i+1$, compute the difference in brightness between foreground image and background image according to a designed brightness threshold, segment each feature in the image.

Step 3 Eliminate the noise contained in the feature block by use of "opening" operation.

Step 4 For each feature block, compute the size, average brightness before image processing and the position and speed of centroid after image processing by clustering algorithm. The size and brightness of feature block are extracted before the "opening" operation.

Step 5 Compute the attribute matching matrix between each feature block in frame $i+1$ and the tracked feature block in frame i in terms of brightness, position, size and speed.

Step 6 Select different weights according to the feature attributes in frame $i+1$ and the relationship between feature blocks, compute the weighted sum of attribute matching matrix.

Step 7 Search the optimized feature correspondence in the weighted sum of attribute matching matrix.

Step 8 Modify the data structure of features according to the feature correspondence. If the feature is isolated, update the position, size, brightness and speed of the feature, else update the position only.

Step 9 If tracking will continue, go to Step 2, else the algorithm is terminated.

4.2.2.3 Feature Attribute Extraction

Feature Segmentation

Before the feature attribute extraction, the segmentation of features is needed. Due to the invariant image background, we can compute the difference in brightness between foreground and background first, and segment each color block via a pre-designed threshold. Because of the similarity in color between maillot color block and background image, the segmented feature block may seem unreasonable. However, automatically setting the threshold is still an open issue, therefore we define a default threshold and provide an interface to allow the user to change the threshold during tracking.

We need to pre-process the segmented features to eliminate some artifacts. The pre-process is in fact "open" operation $OPEN(X)=D(E(X))$ which includes two steps, i. e., "erosion" and "dilation".

Definition 1 Erosion: Translate the structure element B to a, we get Ba. If Ba is included in X, preserve point a, and all the points satisfying above conditions form the Erosion result. Formally described as: $E(X)=\{a \mid Ba \subset X\} = X \ominus B$

Definition 2 Dilation: Translate the structure element B to a, we get Ba. If Ba hits X, preserve point a and all the points satisfying the above conditions form the Dilation result. Formally described as: $E(X)=\{a \mid Ba \uparrow X\} = X \oplus B$

For the sake of simpleness, the same structure element is adopted in erosion and dilation, and the structure element is $B=\{\{0, 0, 0, 0\}, \{0, 0, 1, 0\}, \{0, 1, 1, 0\}, \{0, 0, 0, 0\}\}$.

The pre-processed features may still hold some non-color block region, so we use clustering algorithm and rapid bounding box algorithm to cluster the segmented image regions and get the features enclosed in the bounding box. If obvious difference between the size of clustered feature and that of color block occurs, remove the clustered feature.

Feature Attribute Extraction

The features in image sequence include centroid, speed, size and brightness which are characterized by:

- The position variation of corresponding features in adjacent frames is tiny when the sampling rate is high (above 10 Hz);
- The brightness variation of corresponding features in adjacent frames hardly changes when the motion of features is tiny;
- Suppose the human motion is continuous, the speed and acceleration of features are predictable;
- The size of a feature is invariant when it is not overlapped with other features and there is no continuous motion.

We can build correspondence between the features in adjacent frames according to the characteristics of feature attributes. Therefore, in our approach, we only extract the attributes such as the centroid position, speed, size and brightness.

4.2.2.4 Feature Correspondence

The Quantification of Feature Matching Factor

Currently, the multiple attributes based feature matching algorithm is qualitative, it is hard to integrate multiple attributes together to guide the feature matching. To integrate multiple features, we should quantify the matching between the features in adjacent frames as well as the contribution of each attribute in feature matching. We build a feature attribute matching function and a feature attribute matching factor for each extracted feature attribute. In the attribute matching function, we set T_{att} as a threshold to quantify the matching between the features in adjacent frames. If the difference between the attributes of features is above T_{att}, the features are considered unmatched, otherwise, they can be matched to each other. The attribute matching factor can be inferred through the matching function, which denotes the matching degree of the features in adjacent frames.

The position matching function of the features in adjacent frames can be described as:

$$Dist(\boldsymbol{P}_i^j, \boldsymbol{P}_{i+1}^k) \leqslant T_{\text{positon}} \tag{4-11}$$

where $Dist(\boldsymbol{P}_i^j, \boldsymbol{P}_{i+1}^k)$ denotes the position difference between the jth feature in the ith frame and the kth feature in the $(i+1)$th frame, T_{position} denotes the upper limit of the threshold about position difference, i. e., when the position difference is below T_{position}, the two features are considered matched.

According to equation (4-11), we define the matching factor of feature position as:

$$C_{\text{position}} = 1 - \frac{Dist(\boldsymbol{P}_i^j, \boldsymbol{P}_{i+1}^k)}{T_{\text{position}}} \tag{4-12}$$

When $C_{\text{position}} = 1$, the jth feature in the ith frame is matched with the kth feature in the $(i+1)$th frame in position; when $0 < C_{\text{position}} < 1$, the corresponding two features are nearly matched.

The brightness matching function of the features in adjacent frames can be defined as:

$$\| \boldsymbol{B}_i^j - \boldsymbol{B}_{i+1}^k \| \leqslant T_{\text{bright}} \tag{4-13}$$

where $\| \boldsymbol{B}_i^j - \boldsymbol{B}_{i+1}^k \|$ denotes the brightness difference between the jth feature in the ith frame and the kth feature in the $(i+1)$th frame, T_{bright} denotes the upper limit of the threshold of brightness difference. Only when the brightness difference is below T_{bright}, the two features are possible to match with each other.

According to equation (4-13), we define the matching factor of feature brightness as:

$$C_{\text{bright}}=1-\frac{\| \boldsymbol{B}_i^j-\boldsymbol{B}_{i+1}^k \|}{T_{\text{bright}}} \tag{4-14}$$

When $C_{\text{bright}}=1$, the jth feature in the ith frame is matched with the kth feature in the $(i+1)$th frame in brightness; when $0<C_{\text{bright}}<1$, the corresponding two features are nearly matched.

The speed matching function of the features in adjacent frames can be described as:

$$\| \boldsymbol{V}_i^j-\boldsymbol{V}_{i+1}^k \| \leqslant T_{\text{velocity}} \tag{4-15}$$

where $\| \boldsymbol{V}_i^j-\boldsymbol{V}_{i+1}^k \|$ denotes the speed difference between the jth feature in the ith frame and the kth feature in the $(i+1)$th frame, $\boldsymbol{V}_i^j$ can be obtained according to the position variation of corresponding features in adjacent frames and the time interval, T_{velocity} denotes the upper limit of the threshold of speed difference. Only when the speed difference is below T_{velocity}, the two features are matched with each other.

According to equation (4-15), we define the matching factor of feature speed as:

$$C_{\text{velocity}}=1-\frac{\| \boldsymbol{V}_i^j-\boldsymbol{V}_{i+1}^k \|}{T_{\text{velocity}}} \tag{4-16}$$

When $C_{\text{velocity}}=1$, the jth feature in the ith frame is matched with the kth feature in the $(i+1)$th frame in speed; when $0<C_{\text{velocity}}<1$, the corresponding two features are nearly matched.

The size matching function of the features in adjacent frames can be described as:

$$\frac{\| \boldsymbol{S}_i^j-\boldsymbol{S}_{i+1}^k \|}{\max\ (\boldsymbol{S}_i^j,\ \boldsymbol{S}_{i+1}^k)} \leqslant T_{\text{size}} \tag{4-17}$$

where $\| \boldsymbol{S}_i^j-\boldsymbol{S}_{i+1}^k \|$ denotes the size difference between the jth feature in the ith frame and the kth feature in the $(i+1)$th frame, T_{size} denotes the upper limit of the threshold of size difference. Only when the size difference is below T_{size}, the two features are matched with each other.

According to equation (4-17), we define the matching factor of feature size as:

$$C_{\text{size}}=1-\frac{\| \boldsymbol{S}_i^j-\boldsymbol{S}_{i+1}^k \|}{\max\ (\boldsymbol{S}_i^j,\boldsymbol{S}_{i+1}^k)} \Big/ T_{\text{size}} \tag{4-18}$$

When $C_{\text{size}}=1$, the jth feature in the ith frame is matched with the kth feature in the $(i+1)$th frame in size; when $0<C_{\text{size}}<1$, the corresponding two features are nearly matched.

Feature Matching Matrix

Given the tracking results of the nth frame, feature matching aims to find the feature points in the $(n+1)$th frame which are corresponding to the feature points in the nth frame. In feature matching, only the features with the maximal weighted sum of the attribute matching factor are considered matched. In order to achieve the optimized matching, we first construct the feature matching matrix for 16 feature points extracted from the previous frame which denotes the matching degree of these features to the features in the subsequent frame.

Arbitrary entry S_{ij} in the matrix describes the matching degree of the ith feature in frame n and the jth feature in frame $n+1$ where $S_{ij} \geqslant 0$. When $S_{ij}=1$, the ith feature in frame n is matched with the jth feature in frame $n+1$; when $S_{ij}=+\infty$, the ith feature in frame n is not matched with the jth feature in frame $n+1$.

- When self-occlusion doesn't occur, brightness is the prominent feature in feature matching, and we realize this by increasing the weight factor. When self-occlusion or discontinuous motion occurs, the brightness is less important to feature matching, thus the weight factor drops.
- When few motion features exist in the scene or the motion features do not overlap with each other, the size of the feature plays an important role in feature matching, otherwise, the weight factor of size should be decreased.
- When the video sampling rate is high, the position variation of the features in adjacent frames is very small, thus the features in two frames with similar positions are more probable to be the matching features. When self-occlusion occurs, we can improve the position matching factor to achieve the matching.
- In the high-sampling rate video, suppose the human motion is continuous, then the positions of corresponding features in adjacent frames are predictable. Therefore, when self-occlusion occurs, the problem can be solved by increasing the speed matching factor.

Obtain the attribute matching matrix according to the above methods, and compute the weighted sum of these attribute matching matrices as:

$$\boldsymbol{M}=\boldsymbol{W}_{\text{size}}\boldsymbol{M}_{\text{size}}+\boldsymbol{W}_{\text{bright}}\boldsymbol{M}_{\text{bright}}+\boldsymbol{W}_{\text{velocity}}\boldsymbol{M}_{\text{velocity}}+\boldsymbol{W}_{\text{position}}\boldsymbol{M}_{\text{position}} \quad (4\text{-}19)$$

Each entry in the matrix is:

$$C_{\text{orr}} = \left(\sum_{i=0}^{n} C_i \cdot W_i\right) \Big/ \sum_{i=0}^{n} W_i$$

where W_i is the weighted factor of the ith attribute matching matrix which denotes the contribution of this attribute to feature matching. The weight matrix can be adaptively determined according to whether self-occlusion or discontinuous motion occurs.

Self-occlusion of Feature

We introduce "feature interference" to better describe whether self-occlusion or incontinuous motion occurs. The bounding box of a feature can be described as a 3-tuple: $h=\{height, width, center\}$. The feature interference, which can be described by the bounding box, means the relative position between features. For example, the interference of feature a against feature b along x direction can be described as:

$$Inter_x(a,b)=\frac{a.\text{width}+b.\text{width}}{a.\text{center}.x-b.\text{center}.x}$$

If $\| Inter_x(a,b) \| >2$ or $\| Inter_y(a,b) \| >2$, self-occlusion or discontinous motion occurs between feature a and b. Thus we should adjust the weights of different attribute matching factors in feature correspondence.

Compulsive Constraints in Feature Matching

When the difference of the features in adjacent frames is large, the two features cannot be matched furthermore, and the position constraints between the joints in the human model will be damaged. Therefore, if the attributes of the features in adjacent frames differ greatly from each other, compulsive constraints should be adopted to ensure that the features do not match with each other. The compulsive constraints include:

- The speed v_x and v_y can be obtained according to the feature positions in adjacent frames. If the speed surpasses a certain threshold, the feature points are not matched.
- If the difference in brightness of the features in adjacent frames surpasses the threshold, the features are not matched. We set different thresholds according to the level of brightness.
- If the difference of arbitrary two features in adjacent frames breaks the position constraints between joints of human model, the two feature points are not matched. The position constraints are determined in the first frame.
- If the difference in size of the features in adjacent frames surpasses the threshold, the two feature points are not matched. The size threshold is equal to the size of the features in the first frame.

If arbitrary two features in adjacent frames satisfy the above compulsive constraints, the corresponding attribute matching factor will be assigned the maximal value $+\infty$. In feature correspondence, if there is any entry in the matching matrix equals $+\infty$, the corresponding two features are not matched.

Compute the Maximal Matching

According to equation (4-19), the feature matching matrix of the weighted sum of multiple attributes can be computed as:

$$\boldsymbol{M}=\begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n-1} & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n-1} & A_{2,n} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ A_{m-1,1} & A_{m-1,2} & \cdots & A_{m-1,n-1} & A_{m-1,n} \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n-1} & A_{m,n} \end{bmatrix}$$

where m and n denote the number of features in adjacent frames.

Then, the corresponding features in adjacent frames are computed according to the feature matching matrix. Due to possible self-occlusion or discontinuous motion trajectory, the number of tracked features (n) would not be larger than the number of features in the previous frame (m). Therefore, the feature correspondence should satisfy the constraints as below: in matrix $\boldsymbol{M}$, there exists at least one entry in each column which is the corresponding feature in adjacent frames, while in each row, there is only one entry which is the corresponding feature in adjacent frames. The optimal criterion of feature matching is that the sum of matching factors of m corresponding features reaches the maximum, i. e. , search for all the features (i, j) which satisfy the constrains for feature correspondence, and let equation (4-20) be true.

$$\max\,(sum = \sum_{i=1}^{m} A_{ij}) \quad (j = 1,2,\cdots,n) \tag{4-20}$$

Obviously, the resolution can be obtained by enumeration method from the feature matching matrix, however, the computational complexity is as high as $O(mn)^4$ and this cannot satisfy the need of real-time tracking. Here we adopt a greedy algorithm to solve for the optimal matching feature based on the feature matching matrix.

Step 1 Initialize $R=\varnothing$, $F[m]=0$ (R denotes the optimal feature matching, while F denotes whether the features are matched to each other).

Step 2 Find the entry with the largest value A_{ij} in matrix $\boldsymbol{M}$, enclose (i, j, A_{ij}) in R, $\boldsymbol{M}_i=\boldsymbol{M}$, $F[i-1]=1$.

(1) Remove the ith row and jth column of $\boldsymbol{M}_i$, obtain $\boldsymbol{M}_{ij}$. Find the largest entry A in $\boldsymbol{M}_{ij}$ and enclose (x, y, A) in R, $i=x$, $j=y$, $\boldsymbol{M}_i=\boldsymbol{M}_{ij}$, $F[i-1]=1$;

(2) *If Rank* $(\boldsymbol{M}_i)>1$, go to Step 2;

(3) To arbitrary k which satisfies $F[k]=0$, search for the largest entry A in the ($k+1$)th row of $\boldsymbol{M}$ which is not included in R, the position of this entry is ($k+1$, l), enclose ($k+1$, l, A) in R, $F[i-1]=1$;

(4) If there exists k which satisfies $F[k]=0$, go to Step 5, otherwise algorithm terminates.

4.2.3 Incomplete Motion Feature Tracking Algorithm in Video Sequences

4.2.3.1 Basic Idea

In image-based feature tracking, maximizing the cross correlation between two images and tracking features using optical flow [4] are two main kinds of approaches. Under the assumption that the feature motion is relatively constrained and the brightness is approximately equal for the same feature all the time, optical flow approach matches features according to the criterion of invariable brightness, which is unsuitable for tracking motions that vary greatly [5] or features with discontinuous motion trajectory. On the other hand, attribute-based tracking algorithms firstly extract the attributes of features such as point, line, contour, and then match features between consecutive images by region-based or attribute-based matching approach [6,7], finally the motion trajectory of corresponding features will be built. The difficulty of this attribute-based feature tracking approach lies in extracting feature attributes and quantifying matching function. From the above analysis, we consider that to track incomplete motion features, a new approach must be studied deeply. And we introduce our solution to this problem in this section.

4.2.3.2 Tracking Algorithm

According to the above analysis, the tracking algorithm can be described by the diagram in Fig. 4.6.

Step 1 Select features to be tracked in the first frame. Assuming that no incomplete motion feature exists in the first frame, we match human model with the first frame to obtain joint features and calculate their sizes.

Step 2 Predict the locations of features in frame $i+1$. When the sampling rate of video is high enough (above 10 Hz), we can assume that the human movement is continuous. According to the position and movement vector of features in frame i, the position of each feature in frame $i+1$ can be predicted. The model predicting the position of feature can be described as:

$$\boldsymbol{P}_{i+1}(x,y)=W_{\text{position}}\boldsymbol{P}_i(x,y)+(1-W_{\text{position}})(\boldsymbol{V}_iT+\boldsymbol{P}_i(x,y))$$

where $\boldsymbol{P}_i(x,y)$ is the position vector of feature in frame i, $\boldsymbol{V}_i$ is the speed vector of feature in frame i, T is the time interval between adjacent frames, and W_{position} is the weight factor of position.

Based on the position and size of the tracked features in frame i and the position of the predicted features in frame $i+1$, a search

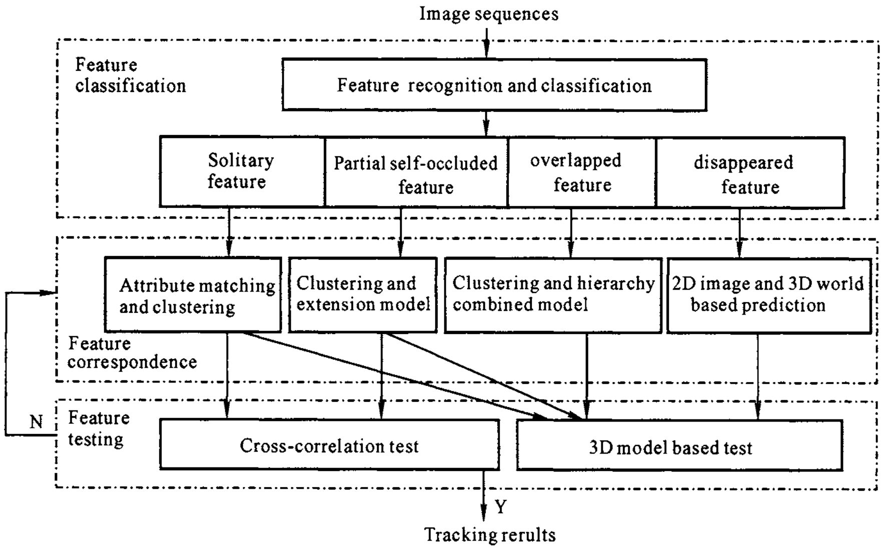


Image sequences
Feature classification
Feature recognition and classification
Solitary feature
Partial self-occluded feature
overlapped feature
disappeared feature
Attribute matching and clustering
Clustering and extension model
Clustering and hierarchy combined model
2D image and 3D world based prediction
Feature correspondence
N
Feature testing
Cross-correlation test
3D model based test
Y
Tracking rerults

Fig. 4. 6 The diagram of tracking algorithm

window W for corresponding features in frame $i+1$ is built. And the pixels with the same attributes are encircled in one window.

Step 3 Calculate the interference degree. To make clear whether there exists self-occlusion or discontinuous motion trajectory of features, the concept of "interference degree" is introduced. Define the bounding box of a feature as a tri-element set $h=\{height, width, center\}$, where *height*, *width*, and *center* are the height, width, and center point of the bounding box respectively. So-called interference degree expressed as bounding box is defined to describe the relation between any two features. For example, the interference degree $Inter_x(a, b)$ in the x direction between feature a and b can be defined as:

$$Inter_x(a,b)=\frac{a.\text{width}+b.\text{width}}{a.\text{center}.x-b.\text{center}.x}$$

If $| Inter_x(a,b) \| > 2$ or $\| Inter_y(a,b) \| > 2$, it means that feature a interfers with feature b.

Step 4 Cluster those features with the same attributes. Clustering algorithm is used to calculate the feature size and feature region. Firstly, in search window W resulted from Step 2,· we utilize sub-block based model matching approach to search for clustering seed point. The color and brightness of seed are approximate to those of corresponding feature in frame i. Then self-adaptive clustering is used to calculate the feature size according to gradient information, and calculate the gravity center of the clustered region.

We use sub-block based matching approach to search for the seed in W, where the sub-block of point p is composed of p and its eight-neighbor points. Firstly, we compute:

$$RedDiff = \sum_{i=0}^{2}\sum_{j=0}^{2}(p[i][j].\mathrm{red} - Feature_m[i][j].\mathrm{red})^2$$

$$GreenDiff = \sum_{i=0}^{2}\sum_{j=0}^{2}(p[i][j].\mathrm{green} - Feature_m[i][j].\mathrm{green})^2$$

$$BlueDiff = \sum_{i=0}^{2}\sum_{j=0}^{2}(p[i][j].\mathrm{blue} - Feature_m[i][j].\mathrm{blue})^2$$

where $p[i][j]$ (i, $j=0,1,2$) denotes the pixels enclosed in the sub-block of point p, while $Feature_m[i][j]$ (i, $j=0,1,2$) denotes the pixels enclosed in the sub-block of point m in the ith frame.

Then, we search for the seed point which is closest to feature point m in the ith frame.

$$\min\,(F(R,G,B)) = w_r \cdot RedDiff + w_g \cdot GreenDiff + w_b \cdot BlueDiff \tag{4-21}$$

$$\| I_p(i+1) - I_m(i) \| \leqslant D \tag{4-22}$$

where $I_m(i)$ denotes the brightness of feature point m in the ith frame, while D is the threshold of brightness. We use adaptive-clustering algorithm to calculate the size of feature and the center of clusters.

Step 5 Classify features. According to the interference degree of the estimated feature and the size of clustered region calculated above, we classify the features to be tracked as solitary features, overlapped features, partially occluded features and disappeared features. Overlapped features mean that one feature overlaps with another feature in the same frame. Partially occluded features are those occluded partially by other parts of human configuration or ambient scenes. Disappeared features are disappearing from current view.

(1) If $\| Inter_x(a,b) \| < 2 \cap \| Inter_y(a,b) \| < 2 \cap \| A - A_0 \| < 0.15A_0$ holds, the feature is solitary feature;

(2) If $\| Inter_x(a,b) \| < 2 \cap \| Inter_y(a,b) \| < 2 \cap \| A - A_0 \| \geqslant 0.15A_0$ holds, the feature is partially occluded feature;

(3) If $\| Inter_x(a,b) \| \geqslant 2 \cup \| Inter_y(a,b) \| \geqslant 2$ holds, the feature is overlapped feature;

(4) If $\| A \| < 0.1A_0$ holds, the feature is disappeared feature.

Step 6 Feature tracking. Select different tracking algorithms according to the classification results of features:

(1) For solitary features, Kalman filter and the feature attribute matching approach can be used, and the tracking result is the center of the clusters;

(2) For overlapped features, the combined templates can be used;

(3) For partial occluded features, the clustering algorithm and the feature extension algorithm can be used;

(4) For disappeared features, the 2D image prediction and the 3D space matching approach can be used.

Step 7 Test tracking results. Cross-correlation testing and 3D model testing are used to test the tracking results. If passes the test algorithm, one feature will be considered as qualified feature. If the feature is unqualified, go to Step 6 and adjust the tracking results until pass the test.

Step 8 If continue tracking, go to Step 2; or else exit.

4.2.3.3 Incomplete Motion Feature Tracking

Main reasons that lead to incomplete motion include: (1) one feature overlaps with others; (2) other parts of human configuration occlude the features; (3) other parts of human configuration make features disappear away from the view. When these cases occur, different strategies will be used in feature tracking.

Tracking Overlapped Features

In motion capture, multiple features overlap with each other, resulting in overlapped features. In this case, the size of clustered region is unequal to that of the feature to be tracked. Thus, clustering algorithm alone cannot track the gravity center. Generally, the movement of limb involves translation and rotation. But other parts rotate only a little, thus only translation is considered in feature tracking. We match the image with the estimated template combined by several overlapped features. The basic procedure is as follows.

Step 1 Estimate motion template for each feature. Motion template can be described by mid-axis, which is defined as $A=\{(x,y),m,n,\theta\}$, where (x,y) is the gravity center of motion template, m, n are the length and width of template, θ is the angle between mid-axis and y-axis which is called rotation angle in this chapter. According to the tracking results in frame i, Kalman filter is used to predict the rotation angle and the gravity center of motion template in frame $i+1$. To calculate the rotation angle in frame i, we assume the warping of feature is mainly caused by rotation circled by z-axis.

In this chapter, we only consider the rotation circled by z-axis. The motion templates of the joints of limbs are described in Fig. 4.7, where O, A, B are the feature templates in frame $i-1$, while O', A', B' are the corresponding feature templates in frame i, and α, Φ are the rotation angles of feature A', B' respectively. The rotation angles can be approximately computed as follows:

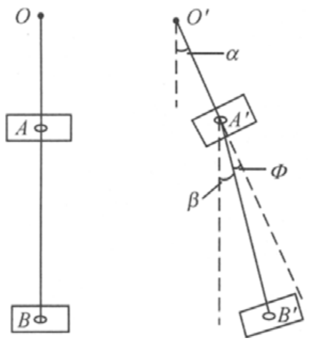


Fig. 4.7 Motion templates

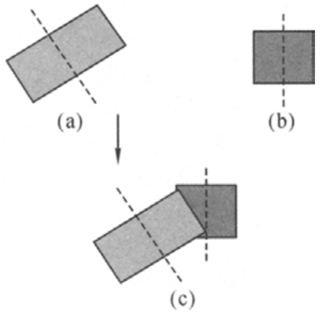


Fig. 4.8 Combined features

$$\Delta x=(x_i(A')-x_i(O'))-(x_{i-1}(A)-x_{i-1}(O))$$

$$\Delta y=(y_i(A')-y_i(O'))-(y_{i-1}(A)-y_{i-1}(O))$$

$$\alpha=2\arctan(\Delta y/\Delta x)$$

$$\Delta x'=(x_i(B')-x_i(A'))-(x_{i-1}(B)-x_{i-1}(A))$$

$$\Delta y'=(y_i(B')-y_i(A'))-(y_{i-1}(B)-y_{i-1}(A))$$

$$\beta=2\arctan(\Delta y'/\Delta x')$$

$$\Phi=\alpha-\beta$$

where $x_i(p)$ and $y_i(p)$ are the coordinate offsets of the gravity center of feature p in the ith frame along x and y directions. According to the orientation and position of the template in the previous frame, we can estimate the warping template of each feature in the current frame.

Step 2 Calculate the combined templates of overlapped features. Let the predicted occluding feature template be $\boldsymbol{I}_1(x,y)$ (Fig. 4.8(a)) and the occluding feature template be $\boldsymbol{I}_2(x,y)$ (Fig. 4.8(b)), we can calculate the combined template (Fig. 4.8(c)) based on interference measures of two features, which can be described as:

$$\boldsymbol{M}_c(x,y)=W(x,y)\boldsymbol{I}_1(x,y)+[1-W(x,y)]\boldsymbol{I}_2(x,y)$$

where $W(x,y)$ is the window function:

$$W(x,y)=\begin{cases}1, & p(x,y)\in A\\ 0, & p(x,y)\notin A\end{cases}$$

where A is the occluding template.

Step 3 Feature matching. In this step, model-based matching is utilized to match overlapped feature with image. If the combined template is $A=\{(x,y), m, n, \theta\}$ and the image block to be matched is

$A'=\{(x', y'), m', n', \theta'\}$, the approach to calculate optimal match is as follows.

(1) If $m \times n < m' \times n'$, $row = m$, $column = n$; or else $row = m'$, $column = n'$.

(2) Grid lines are drawn in the direction arctan θ and arctan$(-1/\theta)$ of block A, and we call the point of intersection sub-pixel X_{ij} ($0 \leqslant i < m$, $0 \leqslant j < n$), then calculate the color X_{ij}[Red], X_{ij}[Green], X_{ij}[Blue] corresponding to the pixels of features in the previous frame;

(3) Calculate Squared Sum of Difference (SSD) of color:

$$E(q) = \sum_{i \in \{r,g,b\}} w_i \iint_C \{\boldsymbol{M}_c^i(x,y) - \dot{\boldsymbol{M}}^i(x,y)\}^2 \mathrm{d}x\mathrm{d}y$$
$$= \sum_{i \in \{r,g,b\}} w_i \iint_C \{W(x,y)\boldsymbol{I}_1^i(x,y) + [1 - W(x,y)]\boldsymbol{I}_2^i(x,y) - \boldsymbol{M}^i(x,y)\}^2 \mathrm{d}x\mathrm{d}y$$

where w_i is the weight factor of color RGB, C is the image region to be matched. Select the region $M(x,y)$ which minimizes $E(q)$ as the optimal matching feature. And calculate the gravity center of each template in the combined template as the tracking results.

Tracking Partial Occluded Features

When the feature disappears from the view or reappearsinto view, features are usually partially occluded by other parts of human configuration. We extend the template of clustered region to track these features more precisely.

Step 1 Clustering algorithm is first used to calculate feature region, and then the size A of clustered region and the gravity center $O(x,y)$ are calculated.

Step 2 Calculate the corresponding rectangle $A(x, y, m, n, \theta)$ with the size equal to the clustered region. The center of rectangle A is the gravity center of clustered region. Its width is the width of template. θ is the rotation angle estimated from the previous frame.

Step 3 Extend the width of rectangle to the width of template along the direction of mid-axis, and then calculate the center of the extended model, which is considered as tracking results. Fig. 4. 9 shows the extended template, where the solid line describes the clustered region with the gravity center O, the corresponding rectangle is denoted by the dotted line, and O' is the gravity center of the extended rectangle. We consider O' as the tracking result. Taking the tracking of two ankle points as example, when right ankle point disappears gradually, the extension of features can be described by Fig. 4. 10.

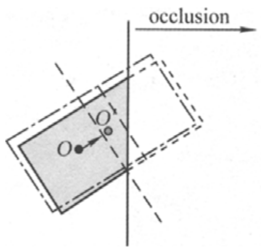


Fig. 4.9 Extended template

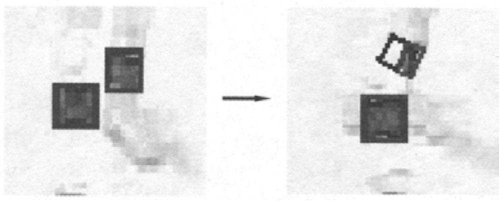


Fig. 4.10 Tracking result of partially occluded feature

Tracking Disappeared Features

Different weight factors for position are used to predict the positions of image features. Then the 3D coordinates of two image features are reconstructed, and the predicted 3D feature is matched to the reconstructed 3D feature for tracking the disappeared feature.

Since the disappeared features will appear in view again, we search for features in the neighborhood of the predicted position. Once a new feature appears in the view, we update the attributes of feature to help the tracking of the next frame.

4.2.3.4 Testing Tracking Results

Error inevitably exists in the tracked results. Experimental results show that once the error in the tracking results is not tested and removed, accumulative errors will become big enough to lead to false match. In order to improve the robustness and precision of the tracking algorithm, we test our tracked results and remove results with big errors. There are two kinds of testing methods, namely the cross-correlation testing and the 3D model based testing adopted in VBHAS V3.0.

Cross-correlation Testing

To test whether the color of the tracking results reflects the color of block correctly, we calculate the correlation coefficient λ based on the brightness of the 3×3 area centered around the feature point:

$$\lambda = \frac{\sum\limits_{i=\pm 2, j=\pm 2} (I_{k-1}(i,j) - \bar{I}_{k-1})(I_k(i-p,j-q) - \bar{I}_k)}{\sqrt{\sum\limits_{i=\pm 2, j=\pm 2} (I_{k-1}(i,j) - \bar{I}_{k-1})^2} \sqrt{\sum\limits_{i=\pm 2, j=\pm 2} (I_k(i-p,j-q) - \bar{I}_k)^2}}$$

where $I_{k-1}(i,j)$ is the brightness of point (i, j) in frame $k-1$, $\bar{I}_{k-1}$ is the mean brightness of the region centered around point (i, j) in frame $k-1$, $I_k(i-p,j-q)$ is the brightness of tracks in frame k, $\bar{I}_k$ is the mean brightness of the region centered around point $(i-p, j-q)$ in frame k. Experimental results demonstrate that it is reasonable to set the threshold as

0. 9. If the correlation coefficient is more than 0. 9, it means the tracks reflect the color of block correctly, or else take p and q which maximize the λ in the neighbor as the tracking results.

3D Model Based Testing

For overlapped features and disappeared features, cross-correlation test cannot be used to test the relationship of brightness between consecutive frames. Instead, we utilize 3D model to test the tracking results. Firstly, we reconstruct 3D coordinates of the corresponding two image features, and then calculate SSD of position between the reconstructed 3D feature and the predicted 3D feature. If SSD is less than the threshold, we take it as qualified feature tracking results.

4.2.4 Human Motion Tracking in Video via HMM

As discussed in previous sections, image attributes alone are not competent for solving all problems of feature tracking. High-level semantic information should be introduced into the feature tracking.

The spatio-temporal prediction and attribute matching based approach can achieve fine tracking results in initial frames (at least 10 frames). Based on this assumption, we propose a Hidden Markov Model (HMM) based tracking approach which does not need many training samples. The training samples are obtained by the motion prediction and attribute matching based approach, and with the tracking going on, more tracking results can be added into the training set. Thus the precision of HMM based tracking can be improved simultaneously. The basic steps can be described as follows.

Step 1 Estimate the candidate point via spatio-temporal correlative model. For the reference point (consider the pelvis joint as the reference point), we use the motion model with evenly inter-frame translation to achieve Kalman filter prediction. For other human configuration joints, we regard the reference point as the benchmark to achieve motion prediction based on motion model with translation and even rotation. The motion model of reference point is:

$$\boldsymbol{S}(k+1) = \boldsymbol{F} \cdot \boldsymbol{S}(k) + \boldsymbol{G} \cdot \boldsymbol{n}(k) \tag{4-23}$$

where

$$\boldsymbol{F}=\begin{bmatrix} 1 & T & \frac{1}{2}T^2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{G}=\begin{bmatrix} \frac{1}{2}T^2 \\ T \\ 1 \end{bmatrix}$$

According to the prediction results, the real tracking result of this point is the reference point of the motion model of other points, which is:

$$\boldsymbol{P}_{i+1}(Child)=\boldsymbol{R}\cdot\boldsymbol{P}_{i+1}(Parent)+\boldsymbol{T}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\omega t & \sin\omega t & 0 \\ -\sin\omega t & \cos\omega t & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} - \boldsymbol{T} \right) + \boldsymbol{T} + \boldsymbol{G}\cdot\boldsymbol{n}(k) \qquad (4\text{-}24)$$

Step 2 Classify the features by Expectation Maximization (EM) based clustering approach and interference degree.

According to the content of Sect. 4. 2. 2, we can compute the interference degree between features. The adaptive EM clustering algorithm is used to obtain the size of feature, denoted as A. And the gravity center of the cluster p can also be computed.

According to the interference degree between features and the size of cluster, we can judge whether the current feature is solitary feature, overlapped feature, partially occluded feature or disappeared feature.

Step 3 For each solitary feature and partially occluded feature, consider the motion and image attribute of candidate point as the observation vector, use motion attribute and image attribute based HMM model to achieve precise feature classification and recognition.

In terms of general HMM model, greedy algorithm with large amount of training samples (Baum-Welch) is suitable for feature tracking in image sequences. We compute the prior model using specific constraints without much training samples.

(1) State vector In HMM based feature tracking, the state vector is designated as the center of the features of each color block. In our VBHAS, the tracking of 16 feature points in each frame is needed. The combination of feature analysis and feature recognition ensures that once the features are recognized, they can be accurately tracked.

(2) The number of observations M corresponding to each state $\boldsymbol{V}=\{v_1, v_2, \cdots, v_M\}$ can be used to describe the observations corresponding to each state, where v_i $(1\leqslant i\leqslant M)$ denotes an observation represented by a set of feature vectors. The dimensionality of the vector is the number of features. In feature tracking, the extracted feature attributes include image attributes and motion attribute. The image attributes include the variation of average brightness of the color blocks in adjacent frames, the variation of average hue and the variation of average saturation; the motion attribute is the inter-frame acceleration a_x and a_y in the image coordinate system. These attributes are independent of each other, and they are only related to the states rather than the time of the states, therefore the dimensionality of the feature vector is five.

Due to the continuous distribution of observations, the total number of observations is hard to obtain. So, we first extract the observation using spatio-temporal constraint based prediction, then according to the experimental result, we select five observations.

(3) The prior model of observation $\boldsymbol{B}=\{b_j(k)\}$ $b_j(k)=P[o_t=v_k \mid q_t=j](1\leqslant k\leqslant M)$ denotes the probability when observation v_k occurs at time t with state j. $\boldsymbol{B}$ is an $N\times M$ matrix, where the row denotes the state and the column denotes the observation. The probabilistic distribution is not obtained by parameter training through greedy algorithm, instead, we suppose each observation satisfies a normal distribution and the parameters are obtained by statistic analysis based on the samples, i. e. , u and σ^2 can be described as:

$$u = \frac{1}{n}\sum_{i=1}^{n} X_i,\ \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - u)^2$$

(4) State transfer probability matrix $\boldsymbol{A}=\{a_{ij}\}$ $\boldsymbol{A}$ is an $N\times N$ matrix, where $a_{ij}=P[q_{t+1}=j \mid q_t=i]$ $(1\leqslant i,j\leqslant N)$, a_{ij} satisfies $a_{ij}>0$ $(\forall i,j)$ and $\sum_{j=1}^{N}a_{ij}=1$ $(\forall i)$. In feature tracking, the prior model instead of the posterior probability obtained by sample training is used in the state transfer probability matrix of HMM. Because the observations are ordered according to y coordinate or the gravity center, the state transfer probability should be inverse ratio to the difference between image coordinates along y directions of gravity center. The y coordinate subjects to the estimated y coordinate in current frame which could be computed according to the position, speed, and acceleration of features in previous frame. State transfer probability matrix can be computed as below:

$$a_{ij}(k)\propto 1/\parallel D_i(k-1)-D_j(k-1)\parallel$$

where $D_i(k-1)=y_i(k-1)+v_i(k-1)\times T+0.5\times a_i(k-1)\times T^2$, $a_{ij}(k)$ is the transfer probability from state i to state j in frame k, $y_i(k-1)$ is the y coordinate of the gravity center of state i in frame $k-1$, $v_i(k-1)$ is the speed offset along y direction of the gravity center of state i in frame $k-1$, a_i $(k-1)$ is the acceleration offset along y direction of the gravity center of state i in frame $k-1$, T is the time interval between adjacent frames.

(5) State probability $\boldsymbol{\pi}=\{\pi_i\}$ State probability means for certain observation $O=(o_1,o_2,\cdots,o_T)$, the probability that certain state occurs at the initiative time. All the π_i in $\pi_i=P[q_1=i]$ $(1\leqslant i\leqslant N)$ construct a $1\times N$ matrix $\boldsymbol{\pi}$. The prior probability instead of posterior probability is designated as the initiative probability of HMM model. The approach for computing the prior of initiative probability is to compute the estimated y coordinate of each state and

make it inversely proportional to corresponding probabilities. This can be formally described as:

$$P_i(k) \propto 1/D_i(k-1)$$

where $P_i(k)$ is the initiative probability of state i in frame k.

Step 4 For each occluded feature and disappeared feature, use motion attribute based HMM model to achieve precise feature classification and recognition.

Step 5 Cross-correlation testing and 3D model based testing are utilized to test the feature.

Given the HMM model described above, we can use Viterbi algorithm to compute the optimal order of states corresponding to the observations. The center of each state is the corresponding tracking result.

4.3 3D Motion Reconstruction

We have discussed in detail the human tracking in multi-view video in the above sections. The tracking result is 2D human motion sequence with incomplete motion information. The result cannot satisfy the requirement of applications such as motion edit and 3D animation, therefore, the reconstruction of 3D motion sequence is necessary. First, we should bridge the gap between spatial points and image feature points via camera calibration, and then 3D reconstruction can be achieved by camera calibration and the tracking results of the image sequences. 3D reconstruction is still an open issue in computer vision[3,8,9].

Before camera calibration and 3D reconstruction, we will introduce the imaging process of objects, i. e. , the projection from 3D spatial points to 2D image plane. Ideal projection model is pinhole model. Fig. 4. 11 describes this procedure, where uO_iv is the 2D image plane, O is the viewpoint, O_iuvz is the camera coordinate system, and $O_wX_wY_wZ_w$ is the spatial coordinate system. Suppose P_u is the ideal projection of object $P_w(X_w, Y_w, Z_w)$ to image plane, while P_d is the practical projection due to the distortion of lens. According to the procedure, calibration is to obtain the relationship between image feature points and 3D spatial points, as well as the inner parameters of cameras. On the contrary, 3D reconstruction aims to recover the 3D coordinates of tracking results via camera calibration and 2D tracking results. We will discuss in detail the approach of 3D reconstruction.

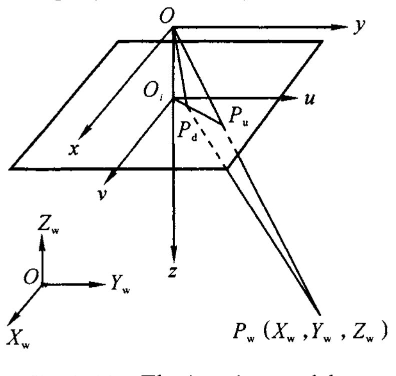


Fig. 4. 11 The imaging model

4.3.1 Tsai Single Camera Linear Calibration Algorithm

Tsai, et al.[10] proposed a rapid and accurate camera calibration algorithm based on former works. According to the observation of calibration model, he proposed a two-step calibration approach: first, the 3D orientation, position and step coefficient are computed; then the focal length, lens distortion coefficient and z coordinate are computed. We will introduce the algorithm in brief.

Following the knowledge of computer graphics, arbitrary point $P_w(X_w, Y_w, Z_w)$ in the world coordinate system can be projected to point $P(u,v)$ on the projection plane through two transformations:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \boldsymbol{R} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} - \boldsymbol{T} \tag{4-25}$$

$$(u,v) = \left(f \cdot \frac{X_c}{Z_c},\ f \cdot \frac{Y_c}{Z_c} \right) \tag{4-26}$$

where

$$\boldsymbol{R} = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix},\ \boldsymbol{T} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix},$$

and (u,v) denotes the image coordinate of corresponding point, f denotes the camera focal length, (X_c, Y_c, Z_c) denotes the coordinate of spatial point under camera coordinate system.

The problem of camera calibration can be described as: given some 3D objects and the corresponding features in 2D image plane, solve for the camera projection parameters $\boldsymbol{R}$, $\boldsymbol{T}$ and f.

Tsai algorithm has two forms: the first is the calibration based on coplanar five points; the second is the calibration based on non-coplanar seven points. And we only discuss the camera calibration algorithm based on non-coplanar seven points.

Firstly, we introduce the intermediate variables $T_y^{-1}s_x r_1$, $T_y^{-1}s_x r_2$, $T_y^{-1}s_x r_3$, $T_y^{-1}s_x T_x$, $T_y^{-1}r_4$, $T_y^{-1}r_5$, $T_y^{-1}r_6$, and the algorithm comprises of two steps.

Step 1 Compute the orientation, position and step coefficient.

(1) Compute the image coordinate (X_f, Y_f). Suppose $s_x = 1$, compute (X_f, Y_f) according to the following equations:

$$X_f = s_x d_x^{-1} X_d + C_x,\ Y_f = d_y^{-1} Y_d + C_y$$

where d_x and d_y are the physical size of each pixel along x and y direction respectively.

(2) Compute $T_y^{-1}s_x r_1$, $T_y^{-1}s_x r_2$, $T_y^{-1}s_x r_3$, $T_y^{-1}s_x T_x$, $T_y^{-1}r_4$, $T_y^{-1}r_5$, $T_y^{-1}r_6$. According to equations (4-25) and (4-26), above equa-

tions can be converted into following linear equations via intermediate variables:

$$[Y_{di}x_{wi}, Y_{di}y_{wi}, Y_{di}z_{wi}, Y_{di}, -X_{di}x_{wi}, -X_{di}y_{wi} - X_{di}z_{wi}] \cdot$$
$$[T_y^{-1}r_1, T_y^{-1}r_2, T_y^{-1}r_3, T_y^{-1}T_x, T_y^{-1}r_4, T_y^{-1}r_5, T_y^{-1}r_6]^T = X_{di} \quad (4\text{-}27)$$

where (X_{wi}, Y_{wi}, Z_{wi}) is the world coordinate of the ith calibration point, while (X_{di}, Y_{di}) is the corresponding image point of the ith calibration point. If the number of corresponding point pairs surpasses seven, $T_y^{-1}r_1$, $T_y^{-1}r_2$, $T_y^{-1}r_3$, $T_y^{-1}T_x$, $T_y^{-1}r_4$, $T_y^{-1}r_5$, $T_y^{-1}r_6$ can be obtained by solving the linear equations.

(3) According to $T_y^{-1}s_xr_1$, $T_y^{-1}s_xr_2$, $T_y^{-1}s_xr_3$, $T_y^{-1}s_xT_x$, $T_y^{-1}r_4$, $T_y^{-1}r_5$, $T_y^{-1}r_6$, compute $(r_1, r_2, \cdots, r_9, T_x, T_y)$. $(r_1, r_2, \cdots, r_9, T_x, T_y)$ can be solved by the following approach:

- Compute $\| T_y \| = (A_5^2 + A_6^2 + A_7^2)^{-1/2}$
- Determine the sign of T_y. Let $A_1 = T_y^{-1}r_1$, $A_2 = T_y^{-1}r_2$, $A_3 = T_y^{-1}r_3$, $A_4 = T_y^{-1}T_x$, $A_5 = T_y^{-1}r_4$, $A_6 = T_y^{-1}r_5$, $A_7 = T_y^{-1}r_6$. The sign can be determined by tentative method: first we select point $I(X_{wi}, Y_{wi}, Z_{wi})$ far away from the center of image and suppose the sign is positive. Then, compute the following values $r_1 = A_1 \cdot T_y$, $r_2 = A_2 \cdot T_y$, $r_3 = A_3 \cdot T_y$, $r_4 = A_5 \cdot T_y$, $r_5 = A_6 \cdot T_y$, $r_6 = A_7 \cdot T_y$, $T_x = A_4 \cdot T_y$, $x = X_w \cdot r_1 + Y_w \cdot r_2 + Z_w \cdot r_3 + T_x$, $y = X_w \cdot r_4 + Y_w \cdot r_5 + Z_w \cdot r_6 + T_y$. If the sign of X is the same as that of X_d, and the sign of Y is the same as that of Y_d, then the sign is positive, or else the sign is negative.
- Compute s_x according to equation $s_x = (a_1^2 + a_2^2 + a_3^2)^{-1/2} \| T_y \|$

(4) Compute the rotation matrix $\boldsymbol{R}$. According to the intermediate variables, we can obtain $r_1 = A_1 \cdot T_y / s_x$, $r_2 = A_2 \cdot T_y / s_x$, $r_3 = A_3 \cdot T_y / s_x$, $r_4 = A_5 \cdot T_y$, $r_5 = A_6 \cdot T_y$, $r_6 = A_7 \cdot T_y$, $T_x = A_4 \cdot T_y$. Due to the orthogonal rotation matrix $\boldsymbol{R}$, the third row can be computed by the cross multiplication of the first row and the second row.

Step 2 Compute the focal length f, the lens distortion coefficient z, and the position T_z. First, we assume the image distortion can be omitted and compute T_z and f according to equatios (4-28) and (4-29).

$$(y_i, -d_yY_i)\begin{pmatrix} f \\ T_z \end{pmatrix} = w_id_yY_i \quad (4\text{-}28)$$

where

$$\begin{cases} y_i = r_4 \cdot x_{wi} + r_5 \cdot y_{wi} + r_6 \cdot z_{wi} + T_y \\ w_i = r_7 \cdot x_{wi} + r_8 \cdot y_{wi} + r_9 \cdot z_{wi} \end{cases} \quad (4\text{-}29)$$

Then, T_z, f, k_1 and k_2 can be precisely computed.

We obtain seven equations based on the seven corresponding point pairs, thus solving for the parameters is to find the solution of the equations. To the coplanar point, suppose the z coordinate of each point is zero, then equation (4-27) is linear equations containing five unknown variables. The calibration parameters can be obtained by solving the linear equations.

4.3.2 Nonlinear and Non-coplanar Calibration Model

In our motion capture system, we improve the Tsai linear calibration model, and propose a novel nonlinear and non-coplanar calibration model. Based on this calibration algorithm, we consider the 3D reconstruction of uncertainty motion and obtain 3D human motion data via nonlinear optimization approach.

Based on the calibration model proposed by Tsai, et al.[10], we adopt nonlinear and non-coplanar calibration model considering the nonlinear distortion of image. The relationship between image feature point (u', v') and spatial point (X_w, Y_w, Z_w) can be described as:

$$Z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1/d_x & 0 & u_0 \\ 0 & 1/d_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \boldsymbol{M} X_w \tag{4-30}$$

$$u = u'(1 + k_1 r^2),\ v = v'(1 + k_1 r^2),\ r^2 = x^2 + y^2 \tag{4-31}$$

$$x = d_x(u - u_0)/S_x,\ y = d_y(v' - v_0) \tag{4-32}$$

where (u, v) is the image coordinate under nonlinear transformation when the nonlinear radial distortion of image is taken into consideration. d_x and d_y are the physical size of each pixel along x and y direction respectively, f is the camera focal length, d_x, d_y and f are inner parameters of the camera. Z_c is the length of spatial point's coordinate under camera coordinate system along z direction. (X_w, Y_w, Z_w) is the position of spatial point under world coordinate system. $\boldsymbol{R} = (R_x, R_y, R_z)$ is a 3×3 orthogonal rotation matrix. $\boldsymbol{t} = (T_x, T_y, T_z)$ is the three-dimensional translation vector. $\boldsymbol{0}^T = (0, 0, 0)$, $\boldsymbol{M}$ is the transformation matrix. k_1 is the radial distortion coefficient with the nonlinear distortion. S_x is the uncertainty proportion factor along x direction under image coordinate system. (u', v') is the observed image coordinate, (x, y) is the image physical coordinate when the uncertainty along x direction is taken into consideration, (u_0, v_0) is the origin position of image physical coordinate system under image coordinate system.

To solve the above equations, the image points and corresponding spatial points are needed, and the calibration object is usually used to calibrate the camera and obtain the 3D coordinates of spatial points.

In our system, the calibration object is designed as Fig. 4.12. The calibration object is a solid frame including 12 spatial points which distribute on different planes and their coordinates can be obtained precisely by metrical instrument. Experiments indicate that the calibration object is competent for multiple applications as long as the error does not surpass 1 cm. The image feature points are automatically tracked based on the camera model.

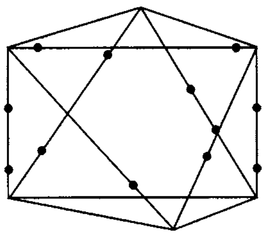


Fig. 4.12 Calibration object

Based on the spatial point (X_w, Y_w, Z_w) and corresponding image feature point (u', v'), we can solve for 11 unknown variables $(f, k_1, u_0, v_0, S_x, R_x, R_y, R_z, T_x, T_y, T_z)$. According to equation (4-30), one point corresponds to two equations, thus six points are necessary to solve for the variables, and usually the number of calibration points are larger than seven.

When solving for the nonlinear calibration model, we first get the initial solutions via the linear calibration model, then the refined solutions can be obtained by nonlinear optimization algorithm. Detailed approach is described as follows.

Step 1 Solve for the initial solution via Tsai linear camera calibration model.

Step 2 Obtain the optimized value of f, k_1, T_z through Levenberg-Marquardtapproach[11].

Step 3 Let the result of Step 2 be the initial input of nonlinear equations, use nonlinear Levenberg-Marquardt method to refine all the parameters except for the center of image plane (u_0, v_0).

Step 4 Let the result of Step 3 be the initial input of nonlinear optimization algorithm, further refine the solutions and obtain the precise calibration parameters.

More attention should be paid to the following two problems.

- In equation (4-30), (u, v) is the image coordinate obtained by nonlinear transformation when the nonlinear radial distortion of image is taken into consideration, while (u', v') is the real tracked image coordinate, therefore equations (4-31) and (4-32) are needed to obtain (u, v) for 3D reconstruction.
- Experimental results indicate that when the origin of world coordinate system is in the view plane of camera, the calibration error generated by nonlinear and non-coplanar camera calibration model is large. In this case, we solve the problem by translating the origin of world coordinate system out of the view plane.

4.3.3 3D Reconstruction of Motion Sequences

3D reconstruction aims to recover 3D motion sequence of the tracked feature points. Traditional approach is to achieve 3D reconstruction via the calibration results. The rationale is to recover 3D information based on the transformation between spatial points and image feature points according to projection principle which can be described as:

$$Z_{C_1} \boldsymbol{u}_1 = \boldsymbol{M}_1 \boldsymbol{X}_p \tag{4-33}$$

$$Z_{C_2} \boldsymbol{u}_2 = \boldsymbol{M}_2 \boldsymbol{X}_p \tag{4-34}$$

where Z_{C_1} and Z_{C_2} are z coordinates of spatial point under two cameras' coordinate system respectively, $\boldsymbol{X}_p$ is the homogeneous coordinate of spatial point P under world coordinate system, $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are the homogeneous coordinates of P_1 and P_2 which are the projections of spatial point P onto the image planes, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are projection matrices of two cameras respectively.

When the tracking result (u', v') is known, six equations are enough for solving for five unknown variables (Z_{C_1}, Z_{C_2}, $\boldsymbol{X}_p(x, y, z)$). The detailed approach is to obtain (u, v) via equations (4-31) and (4-32) first, then equations (4-33) and (4-34) can be transformed to get:

$$\boldsymbol{A}\boldsymbol{X}_p = \boldsymbol{B} \tag{4-35}$$

where $\boldsymbol{A}$ is 4×3 matrix, and $\boldsymbol{B}$ is 4×1 matrix. Solve for $\boldsymbol{X}_p$ through least square algorithm and obtain:

$$\boldsymbol{X}_p = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B} \tag{4-36}$$

The least square algorithm ensures the minimal distance between the recovered 3D point and the two lines connecting the origins of camera coordinate system and image feature points (one line corresponds to one camera).

Experimental results show that the approach can satisfy requirements of most applications, however, when we project the reconstructed spatial feature points back onto the image plane, the error tends to be large. As Fig. 4. 13 shows, p is the reconstructed spatial point corresponding to the image feature points p_1 and p_2, while p_1' and p_2' are the projections of p onto the image planes. p_1' and p_2' deviate from p_1 and p_2 significantly, and this is intolerable to feature tracking. This is because uncertainty is inevitable in 3D reconstruction. For example, the error of calibration result makes the two lines connecting the origins of camera coordinate system and image feature points un-intersectant. Least square algorithm only generates the approximate solution, moreover, there is no physical meaning in solving the problem through least square algorithm. The projection of reconstructed spatial point is often used to guide the tracking,

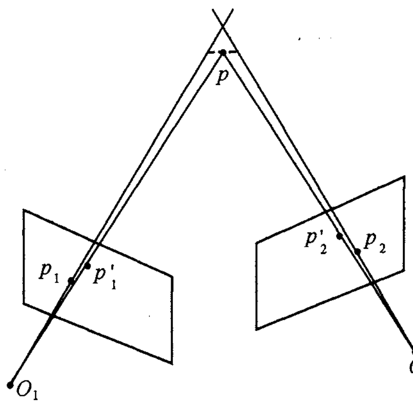


Fig. 4. 13 Image feature points and reconstructed 3D spatial point

but the above approach cannot satisfy our need. If the projections of reconstructed spatial point approach the tracking results, the requirement can be satisfied, thus we consider the 3D human motion reconstruction approach which takes motion uncertainty into consideration. The strategy and realization of our approach are described as follows.

The optimization criterion of our approach is to minimize the distance between the tracked image feature points and the image projections of the reconstructed 3D spatial point, i. e. , minimizing the following formula:

$$d = \sum_{i=1}^{2}\left[\left(u_i - \frac{\boldsymbol{M}_{11}x + \boldsymbol{M}_{12}y + \boldsymbol{M}_{13}z + \boldsymbol{M}_{14}}{\boldsymbol{M}_{31}x + \boldsymbol{M}_{32}y + \boldsymbol{M}_{33}z + \boldsymbol{M}_{34}}\right)^2 + \left(v_i - \frac{\boldsymbol{M}_{21}x + \boldsymbol{M}_{22}y + \boldsymbol{M}_{23}z + \boldsymbol{M}_{24}}{\boldsymbol{M}_{31}x + \boldsymbol{M}_{32}y + \boldsymbol{M}_{33}z + \boldsymbol{M}_{34}}\right)^2\right] \tag{4-37}$$

where (x,y,z) is the coordinate of spatial point. Compute the partial derivatives of x, y, z and set the derivatives to be zero to ensure d reach its minimal value.

$$\begin{cases} f_1(x,y,z) = \partial d/\partial x = 0 \\ f_2(x,y,z) = \partial d/\partial y = 0 \\ f_3(x,y,z) = \partial d/\partial z = 0 \end{cases} \tag{4-38}$$

The nonlinear equations in (4-38) can be solved by Newton method for the coordinate of spatial point (x,y,z).

4.4 Case Studies: VBHAS V3.0

VBHAS V3. 0 is an intelligent animation system which integrates camera calibration, video decoding, image synchronization, 2D feature tracking, 3D motion reconstruction, motion edit, 3D modeling and motion reuse. The system framework can be described as Fig. 4. 14, and the main user interface can be seen in Fig. 4. 15.

Two-camera-based human animation technique includes following steps.

Step 1 Decoding, denoise and synchronization algorithms are used to preprocess the raw video sequences.

Step 2 The human motion features are then captured by the automatic motion tracking algorithm, and the result is the 2D human motion sequence.

Step 3 Reconstruct 3D human motion sequence through the camera calibration result and 3D reconstruction algorithm, then the reconstructed 3D motion is denoised to obtain smoothened 3D motion sequence.

The motion synthesis function of VBHAS V3. 0 is composed of follow-

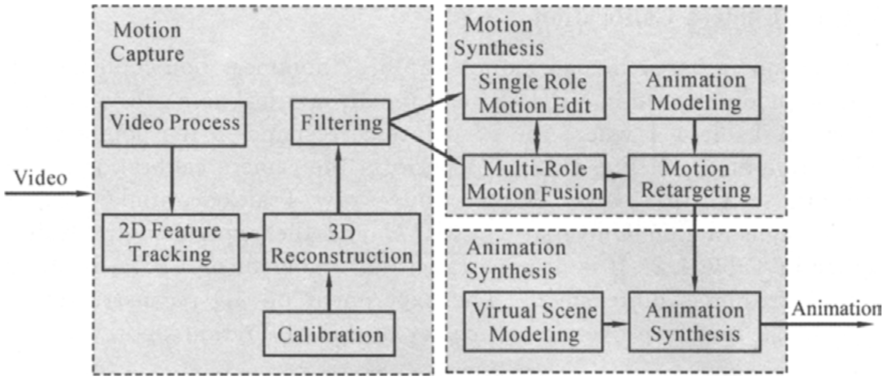


Fig. 4. 14 The framework of VBHAS V3. 0

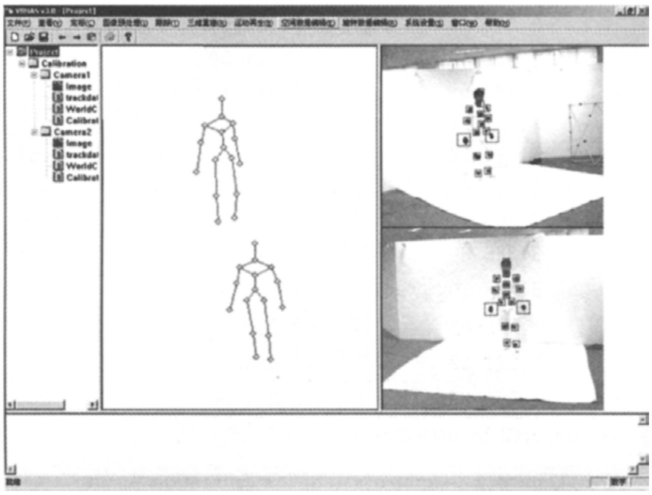


Fig. 4. 15 The main interface of VBHAS V3. 0

ing modules:

- 3D motion editing through motion editing and reuse algorithm, resulting in motion sequence which can satisfy the requirement of the scene or the animator;
- Multiple characters' motion fusion according to the captured human motion and the result of motion editing;
- Create vivid personalized animation by transforming the edited and fused 3D motion sequence to the pre-built 3D animation model. In animation synthesis, complex virtual scene is created and combined with the motion to help the animator build multiple personalized animations.

We will discuss camera calibration, feature tracking and motion reconstruction in detail in the following sections.

4.4.1 Camera Calibration

In two-camera-based human motion capture, nonlinear non-coplanar calibration model is used in VBHAS, and the 3D metrical data of the model is shown in Table 4.1 where the 12 points correspond to the points of the calibration object in Fig. 4.12. We perform the camera calibration on the test data set (http://www.cs.cmu.edu/~rgw/TsaiCode.html) provided by Carnegie Mellon University (CMU), and the experimental result is shown in Table 4.2. It is demonstrated that the error of VBHAS calibration algorithm is quite small. The experiment on 3D reconstruction in Sect. 4.4.3 further proves the efficiency of this calibration algorithm.

Table 4.1 Metrical data of calibration object

Num.	X/mm	Y/mm	Z/mm
1	2,040.16	2,081.67	1,852.70
2	3,041.40	2,074.50	1,856.16
3	3,289.57	2,076.10	1,517.80
4	1,787.88	2,084.50	1,594.40
5	1,785.70	2,084.52	1,278.50
6	2,117.08	1,715.44	1,103.69
7	2,664.60	1,702.75	1,077.69
8	3,285.89	2,076.10	1,251.30
9	2,568.10	2,508.52	1,568.44
10	2,852.88	2,355.39	1,206.30
11	2,221.62	2,519.37	1,585.90
12	1,925.37	2,244.96	964.25

* Remark is measured by instrument.

Table 4.2 Comparison of the calibration result

Type	T_x/ mm	T_y/ mm	T_z/ mm
Ideal data	−100.268888	−85.323235	1999.761873
Our result	−100.088440	−85.136185	1999.154367
Error	0.180448	0.18705	0.607506
Type	R_x (radian)	R_y (radian)	R_z (radian)
Ideal data	0.52348854	0.017487199	0.03500024
Our result	0.523401	0.017581	0.034999
Error	0.00008754	0.000094	0.00001

4.4.2 Feature Tracking

4.4.2.1 Tracking Result of VBHAS V3.0

To validate the efficiency of the multi-view feature tracking algorithm proposed in this chapter, we apply feature tracking to the video sequence containing *jumping* motion. First the tracking result of optical flow approach is presented then the comparison of optical flow tracking with the feature tracking approach is given.

Fig. 4. 16 describes the tracking results by optical flow algorithm (the white points in the color block are the tracking results). Fig. 4. 16 indicates that to a video sequence of human motion containing 25 frames: (1) When the distance between human features is small, the inter-feature influence is large and the tracking results are not good enough. As is shown in Fig. 4. 16(a) and (b), when the distances between pelvis point, right hip point and left hip point in the 10th, 15th, 20th, 25th frames are small, the error of feature tracking tends to be large, while the tracking results of the solitary features are acceptable; (2) When the motion of human features is salient, the feature tracking tends to fail, e. g. , the right elbow point and the left elbow point of the 20th, 25th frames in Fig. 4. 16 (a) and (b). Experimental results indicate that optical flow algorithm is incompetent for non-solitary features and features with salient motion, therefore, optical flow algorithm is not suitable for complex problems such as human motion tracking.

Fig. 4. 17 describes the tracking results by attribute quantization approach. We can observe that: (1) the tracking of most feature points are

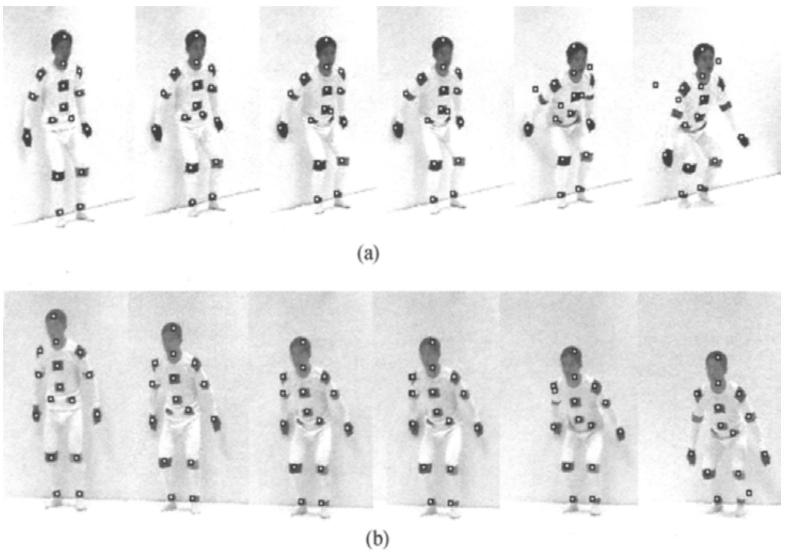


Fig. 4. 16 Tracking results by optical flow algorithm. (a) The 1st, 5th, 10th, 15th, 20th, 25th frame captured by the left camera and the tracking results; (b) The 1st, 5th, 10th, 15th, 20th, 25th frame captured by the right camera and the tracking results

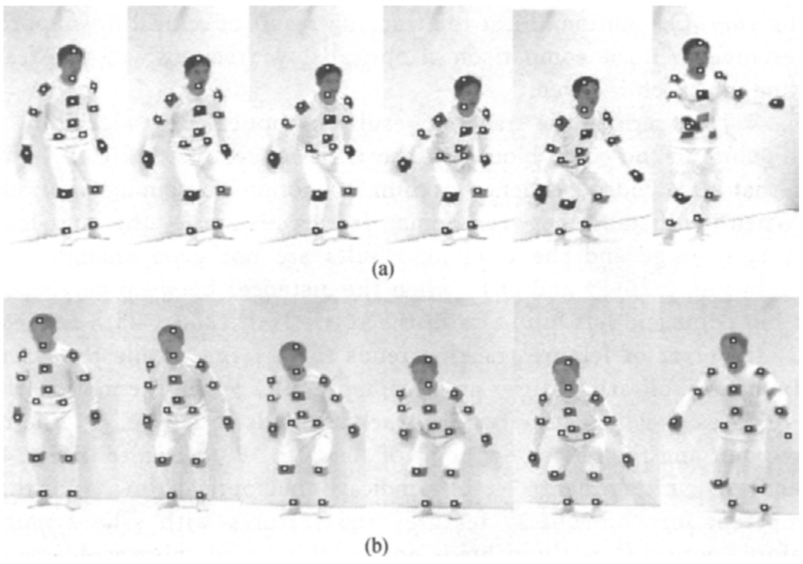


Fig. 4. 17 Tracking results by the attribute quantization approach. (a) The 1st, 5th, 10th, 15th, 20th, 25th frame captured by the left camera and the tracking results; (b) The 1st, 5th, 10th, 15th, 20th, 25th frame captured by the right camera and the tracking results

quite accurate; (2) the right wrist point, pelvis point, right hip point and left hip point in the 25th frame of Fig. 4. 17(a) overlap with each other, and in Fig. 4. 17(b), the tracking of right hip point and left elbow point in the 15th frame fails. This is because the tracking results of our approach are always located at the center of color blocks, and when error occurs in the extraction of color blocks (the features overlap with each other or the color of background is similar to that of the color block), the feature matching will suffer from inaccuracy. The experimental results indicate that when the features do not overlap with each other and the contrast between the color blocks and the background is significant, the tracking can be successfully performed by this approach.

Fig. 4. 18 shows the tracking results of incomplete motion feature tracking approach which adopt the spatio-temporal correlation based method. To validate that the approach is suitable for more complex motion, we apply the approach to jumping motion sequence with 60 frames. From Fig. 4. 18(a) we observe that only the tracking of the right hip point in the 25th frame and the right wrist point in the 20th frame is not accurate, while in Fig. 4. 18(b), all the feature points except for the left knee point in the 30th frame can be accurately tracked. Experimental results indicate that the approach is superior to the two kinds of tracking approaches introduced above.

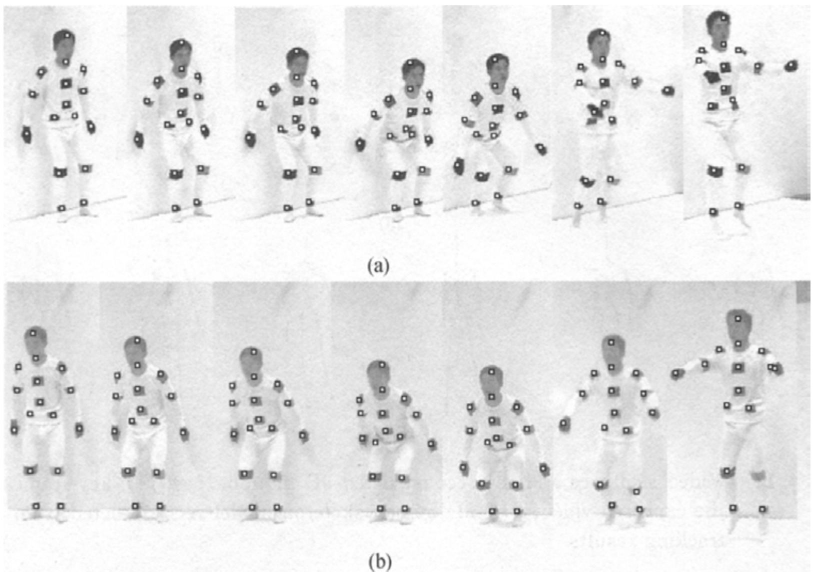


Fig. 4. 18 Kalman filter based the spatio-temporal tracking results. (a) The 1st, 5th, 10th, 15th, 20th, 25th, 30th frame captured by the left camera and the tracking results; (b) The 1st, 5th, 10th, 15th, 20th, 25th, 30th frame captured by the right camera and the tracking results

The above experimental comparisons are based on the 2D tracking results, we will then perform 3D reconstruction on the 2D tracking results and validate the algorithm efficiency by analyzing the vividness of 3D motion sequence. Fig. 4. 19 depicts the captured video sequence of human walking and the reconstructed 3D human motion sequence. Self-occlusion occurs on hip point and wrist points on the 3rd frame in Fig. 4. 19(a). Fig. 4. 19(b) shows the experimental results in which the tracking is conducted by the attribute quantization based approach and the reconstruction is achieved by computer vision technique. The experimental results of 3D reconstruction indicate that the approach proposed in this chapter can achieve accurate motion tracking even though self-occlusion occurs in human motion.

4.4.2.2 Performance Analysis

The human tracking results of several approaches have been presented in the previous sections, now we will analyze in detail the performance of the approaches when tracking certain feature points. We only discuss the performance of attribute quantization based tracking approach. Specifically, we compare the tracking results of left hand point and left hip point of a human walking sequence. The frame rate of camera is 25 fps, the background is simple and the motion of arms is significant so that the left hand

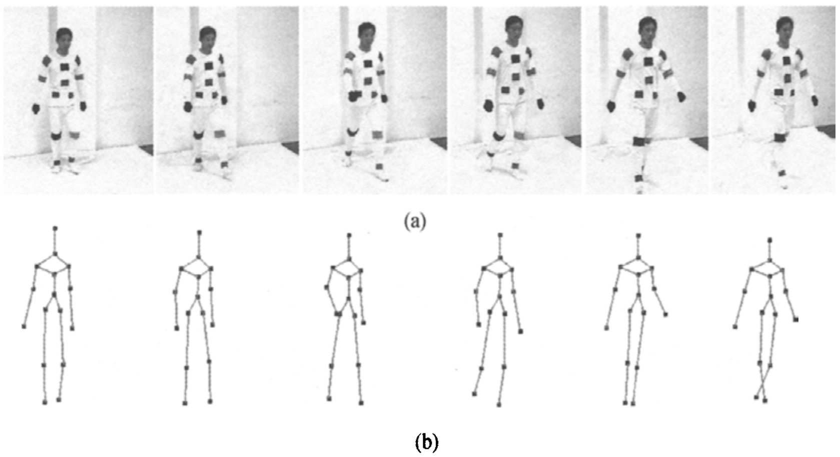


Fig. 4. 19 Video sequence and the reconstructed 3D motion. (a) The key-frames of the captured video; (b) 3D human skeleton model reconstructed from the tracking results

point and left hip point are occluded by human configuration some times. Two different approaches are executed for this experiment: (1) image-based tracking approach, represented by optical flow algorithm; (2) our proposed attribute quantization based approach. The tracking result in Fig. 4. 19 is used to analyze the performance, and the comparison of these two approaches is shown in Fig. 4. 20. In video-based motion capture, the weight factors of brightness, position, speed and size of the solitary features are 0. 4, 0. 1, 0. 4 and 0. 1 respectively. To non-solitary features such as pelvis point, hip points and wrist points, the weight factors of position, speed and brightness are 0. 3, 0. 5 and 0. 2 respectively.

Fig. 4. 20 indicates that: (1) when self-occlusion occurs on left hand point and left hip point, they cannot be correctly tracked, because optical flow algorithm degenerates greatly due to significant change of brightness; (2) when two features are correlated to each other and thus lead to self-oc-

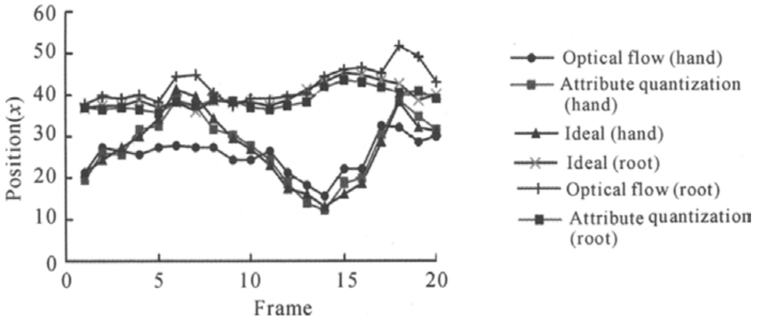
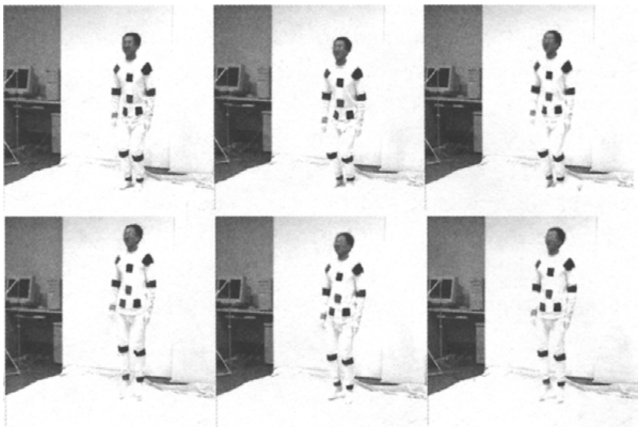
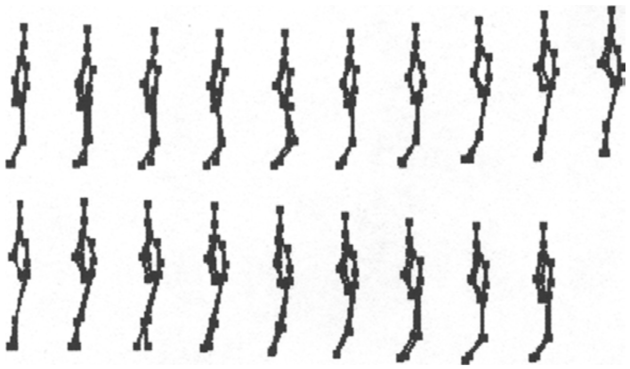


Fig. 4. 20 The curves of human walking obtained by different approaches



(a) The 1, 4, 8, 12, 16, 19th frames captured by one camera



(b) The 1st to 19th frames of reconstructed 3D human skeleton in side view

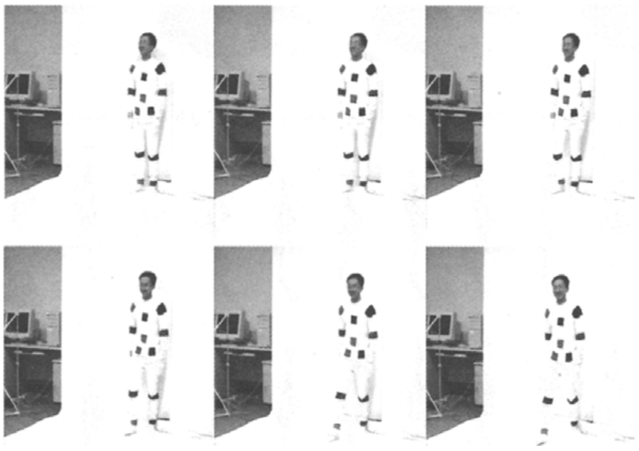
Fig. 4. 21 3D reconstruction of jumping motion. (a) Several key-frames of the captured high jump video sequence; (b) The corresponding reconstructed 3D human motion sequence

clusion, multi-attribute quantization based approach can still obtain fine tracking results; (3) when the appearance of the feature point changes from self-occlusion to non self-occlusion, the multi-attribute quantization based approach bridges the gap between different feature points. While the brightness based approach degenerates rapidly when the appearance of feature points changes continually. Therefore, the proposed attribute quantization based approach considers the contributions of multiple feature attributes, and improves the tracking results of self-occluded motion sequence.

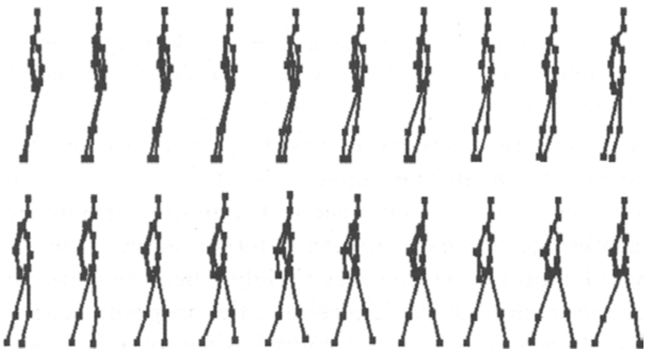
4. 4. 3 3D Reconstruction

Two CCD cameras are utilized in VBHAS V3. 0 to capture human motion sequences. Some experimental results of two-camera-based 3D human motion reconstruction are shown in Figs. 4. 21 and 4. 22. Fig. 4. 21 shows the reconstructed 3D motion sequence on some key frames of a jumping video. Fig. 4. 22 is an example on walking video.

Fig. 4. 23 shows the walking sequence captured by one camera and the corresponding reconstructed 3D human motion sequence in frontal view and side view respectively. We can observe that: the captured 3D human motion is reasonable and realistic human animation can be synthesized by retargeting the captured motion to pre-built 3D human model. These experimental results demonstrate the feasibility and validity of the human motion capture approach introduced in this chapter.



(a) The 1, 4, 8, 12, 16, 20th frame captured by one camera



(b) The 1to20 frame of reconstructed 3D human skeleton in side view

Fig. 4. 22 Reconstruction of walking motion. (a) Several key-frames of the captured walking video sequence; (b) The corresponding reconstructed 3D human motion sequence

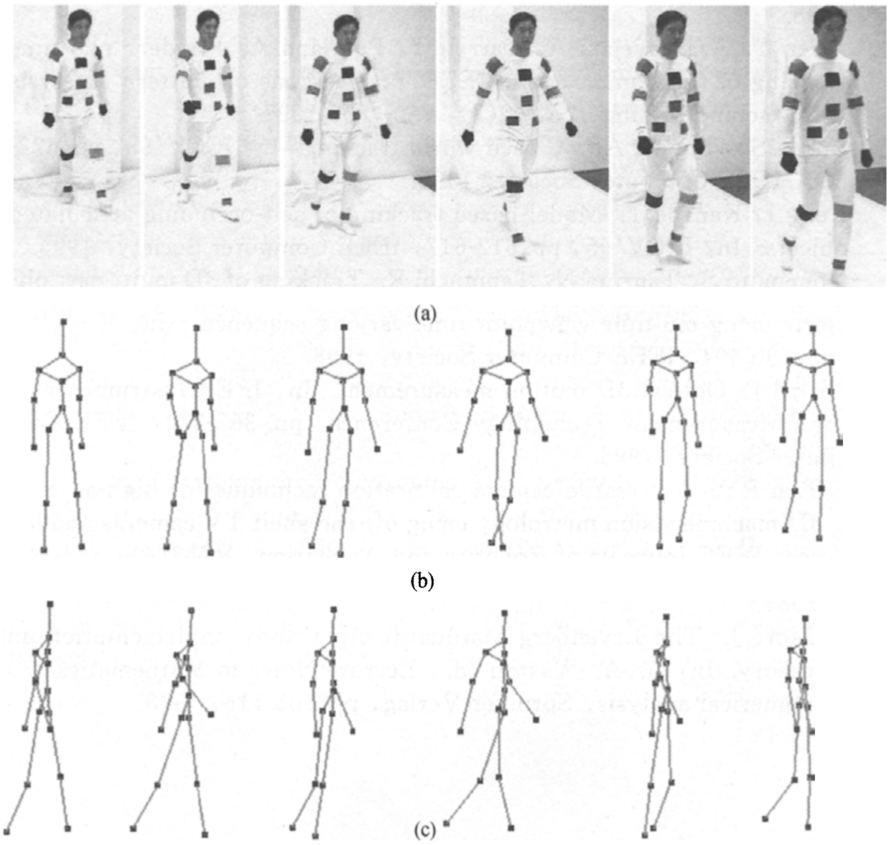


Fig. 4. 23 Key-frames and the captured 3D human motion sequence. (a) The key-frames captured by one camera. (b) The reconstructed 3D human motion sequence in frontal view. (c) The reconstructed 3D human motion sequence in side view

References

1. Kakadiaris A, Metaxas D. 3D human body model acquisition from multiple views. In: ICCV'95, pp. 618-623, IEEE Computer Society, 1995.
2. Liu X, Zhuang Y, Pan Y. Video based human animation technique. In: ACM Multimedia, pp. 353-362, ACM Press, 1999.
3. Bregler C, Malik J. Tracking people with twists and exponential maps. In: CVPR'98, pp. 8-15, IEEE Computer Society, 1998.
4. Fua P, Gruen A, Plankers R, Apuzzo N, Thalmann D. Human body modeling and motion analysis from video sequences. International Archives of Photogrammetry and Remote Sensing, 32(B5): 866-873,

1998.
5. Wren C, Azarbayejani A, Darrell T, Pentland A. Pfinder: real-time tracking of the human body. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7): 780-785, 1997.
6. Intille SS, Bobick AF. Closed world tracking. In: ICCV'95, pp. 672-678, IEEE Computer Society, 1995.
7. Rehg J, Kanade T. Model-based tracking of self-occluding articulated objects. In: ICCV'95, pp. 612-617, IEEE Computer Society, 1995.
8. Yonemoto S, Tsuruta N, Taniguchi R. Tracking of 3D multi-part objects using multiple viewpoint time-varying sequences. In: ICPR'98, pp. 490-494, IEEE Computer Society, 1998.
9. Sabel J. Optical 3D motion measurement. In: IEEE Instrumentation and Measurement Technology Conference, pp. 367-370, IEEE Computer Society, 1996.
10. Tsai RY. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. IEEE Journals of Robotics and Animation, RA-3(4): 323-344, 1987.
11. More J. The Levenberg-Marquardt algorithm: implementation and theory. In: G. A. Waston ed., Lecture Notes in Mathematics 630: numerical analysis, Springer-Verlag, pp. 105-116, 1978.

5

Video-based Facial Animation Techniques

Facial expression creation is an indispensable part of human animation. As a special branch of computer graphics and image processing, facial expression synthesis has been an attractive topic for over 30 years and has appeared in many applications, e. g. , complex human-computer-interface, interactive game, multimedia teaching, remote experience of virtual reality, and computer animation. Facial animation techniques have evolved from traditional key-frame technique to current techniques of image warping, video-based animation, physics-based animation and behavior animation. Related research domain includes computer graphics, artificial intelligence and machine learning. The requirement of generating high quality facial animation is more and more urgent. The traditional literalness and rigid facial animation cannot satisfy people's need, thus it is necessary to develop high-quality and high-efficiency facial animation techniques.

Due to the technology progress and cost reduction, plenty of digital equipments appear in everyday life. Under this background, the digital video has become an important information source in people's life, thus video-based face modeling and expression synthesis is a trend of facial animation techniques. Investigation indicates that over 83% input information of human brain comes from visual information, so it is important to explore the abundant information in digital videos and develop novel face modeling and facial animation techniques. This chapter introduces the main problems occurring in facial animation creation and new techniques to solve these problems. The new techniques are closely related to the state of the art theories and approaches of artificial intelligence and aim to guide the authors to observe and think in a new manner. The following content will focus on video-based facial expression synthesis, facial feature tracking, 3D reconstruction of tracking data, video-based specific face modeling and data driven facial animation, respectively.

5.1 Facial Expression Hallucination

Facial expression hallucination is a novel kind of facial expression synthesis technique, whose objective is to automatically estimate various facial expressions according to the given face image with neutral expression. Some latent applications include human-computer interaction, film making and game entertainment. However, due to the sensitivity to human face, facial expression hallucination is still a challenging topic in computer vision.

Current facial expression synthesis techniques can be divided into two classes, i. e. , image-based and 3D model-based. Liu, et al.[1] proposed an expression mapping approach based on Expression Ratio Image (ERI). Combining illumination changes of one's expression with geometry warping, they mapped an ERI to arbitrary face and generated more expressive facial expressions. Noh and Neumann[2] proposed an expression clone algorithm, which passes the motion vectors of the vertices in source face model to that in target face model. The two approaches retarget the source facial expression to a new object, but the synthesized facial expression is just the simple simulation of the source expression. Zhang, et al.[3] developed a geometry-driven facial expression synthesis system which could generate photorealistic facial expressions through blending sub-region texture images according to the facial feature positions. This technique is based on image warping and is not feasible when only one target neutral face image is given. Pyun, et al.[4] developed an example based method. They transformed facial expression to a target face through blending the pre-defined face models corresponding to the example source face models. However, this needs toilsome works to build the target key face models for blending. In this chapter, we introduce our facial expression hallucination technique, both image-based and video-based. The video-based facial expression hallucination technique is an extension of image-based one, and can be regarded as innovative work in pattern recognition and computer vision domain.

5.1.1 Image-based Facial Expression Hallucination

In this section, we introduce an image-based facial expression hallucination approach. The purpose is shown in Fig. 5. 1. We learn the relationship between neutral face images and expressional ones with the help of a facial expression database. Then given a neutral face image that is not in the database, we could infer the plausible and expressive expressions. For each kind of facial expressions, we can set up an independent facial expression database. The algorithm and process are all the same when hallucinating different kinds of expressions. For simplicity, we only handle the happy expression.

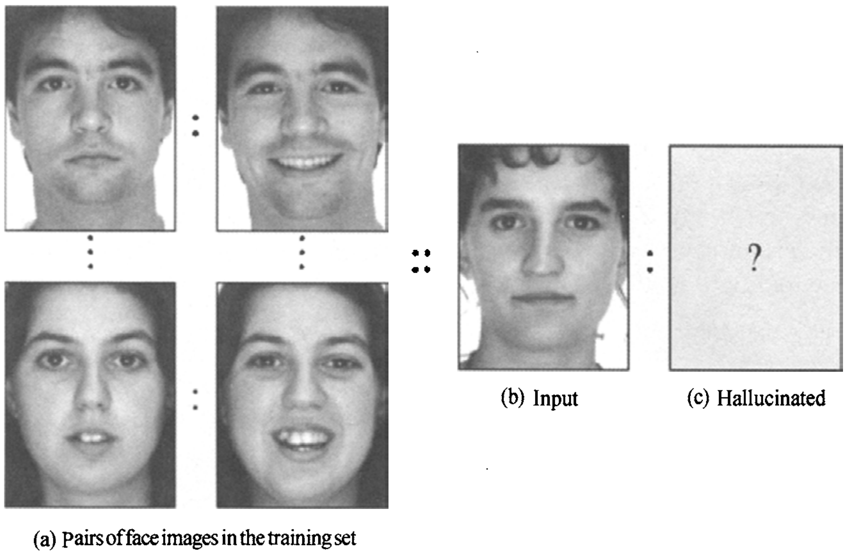


Fig. 5. 1 Hallucinate the plausible facial expression of an input neutral face image with the help of training set

There are two steps to hallucinate facial expression: the first one is global manifold learning and inference, and the second one is local patch-based belief propagation. The happy face image generated by global method looks smooth, which we take as the preliminary results. To make it more expressive, local low-level learning is carried out to do further refinement for the residual face image.

5.1.1.1 Face Image Synthesis by Manifold Learning and Inference

The global face image synthesis consists of the following two parts.

Learning step: (1) Learn the subspace (manifold) of face images with neutral expression in database. Then we obtain the intrinsic parameters of that manifold. Therefore, we can reconstruct face images by the parameters; (2) Use the similar method to learn the manifold of face images with happy expression in database; (3) Learn a relationship of these two manifolds. Because the face images in the training set are in pairs, the relationship can be obtained by a multi-variate regression.

Inference step: (1) Obtain the parameters of an input face image with neutral expression; (2) Infer its happy parameters from the relationship between learned neutral and happy intrinsic parameters; (3) Reconstruct the happy face image by the happy parameters.

Dimensionality Reduction

A face image can be regarded as a point in a high-dimensional vector space, with each dimension corresponding to the gray-scale of one pixel in the image. For a 96 × 128 image, the dimensionality p is as high as 12,288 and apparently very high. In order to get parameters to describe the face image, the conventional method is Principal Component Analysis (PCA) or Eigenface. However PCA only discovers the structure of data lying on a linear subspace of the high-dimensional vector space. It is questionable for Eigenface to presume that all face images (e. g. with neutral expression) form a hyperellipsoid cloud. It is more proper to assume that face images with certain expression are lying on a nonlinear subspace, which can be represented by a nonlinear manifold.

Nonlinear Manifold

A manifold is a topological space which is locally Euclidean. The local Euclidean property means that there is a neighborhood which is topologically the same as the open unit ball in $\mathbf{R}^m$.

We use ISOMAP [5] to carry out nonlinear dimensionality reduction for face images. We assume that face images with a certain facial expression lie on an unknown manifold M embedded in the high-dimensional observation space I. Let I_i denote the coordinate of the ith observation. We seek a mapping $f: I \rightarrow Y$ from the observation space I to a low-dimensional Euclidean feature space Y that preserves the intrinsic metric structure of the observations as much as possible.

The reverse mapping $f^{-1}: Y \rightarrow I$ from abstract facial feature vectors to images can be learned by fitting a regularization network to the corresponding points in both spaces. Dimensionality reduction using ISOMAP can not only capture more meaningful structural information than PCA, but also has much smaller residual variance in face image than PCA with the same dimensionality; therefore ISOMAP is better than PCA in dimensionality reduction.

Obtain Parameters for Neutral Face Images

For a set of N neutral face images, $\{\boldsymbol{I}_i^{\text{nt}}\}_{i=1}^{N}$, $\boldsymbol{I}_i^{\text{nt}} \in \mathbf{R}^D$ with $N<D$, suppose we already have obtained the set of pairwise geodesic distances $\delta_{ij} = d(\boldsymbol{I}_i^{\text{nt}}, \boldsymbol{I}_j^{\text{nt}})$. We need find the neutral facial parameters embedded in the low-dimensional manifold $\mathbf{R}^m$.

Given these geodesic distances, we construct a matrix $\boldsymbol{A} = [a_{ij}]$ and $a_{ij} = -\delta_{ij}^2/2$. Defining $\boldsymbol{H}$ as the center matrix, $\boldsymbol{H} = \boldsymbol{I}_N - \boldsymbol{e}\boldsymbol{e}^{\text{T}}/N$, where $\boldsymbol{e}$ is the N-dimensional vector of all ones, we set $\boldsymbol{B} = \boldsymbol{HAH}$. Let the eigendecomposition of $\boldsymbol{B}$ be: $\boldsymbol{B} = \boldsymbol{V\Lambda V}^{\text{T}}$, where $\boldsymbol{\Lambda}$ is a diagonal matrix with eigenvalues and $\boldsymbol{V}$ is the $N \times p$ matrix whose columns are the eigenvectors of $\boldsymbol{B}$. Since $N<D$, there are $D-N$ zero eigenvalues in $\boldsymbol{\Lambda}$. If the eigenvalues are ordered as $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \lambda_D \geqslant 0$, then $\boldsymbol{B} = \boldsymbol{V}_N \boldsymbol{\Lambda}_N \boldsymbol{V}_N^{\text{T}}$, where $\boldsymbol{\Lambda}_N = \text{diag}\ (\lambda_1, \cdots, \lambda_N)$ and $\boldsymbol{V}_N$ is the $N \times D$ matrix whose columns correspond to the first N eigenvectors of $\boldsymbol{B}$.

The coordinates $\boldsymbol{Y}_i^{\mathrm{nt}}$ are set to the top m eigenvectors of $\boldsymbol{V}_N$. In detail, let $\boldsymbol{v}_d^i$ be the ith component of the dth eigenvector $\boldsymbol{v}_d$ of matrix $\boldsymbol{V}_N$. Then we set the dth component of the m-dimensional coordinate vector of $\boldsymbol{Y}_i^{\mathrm{nt}}$ equal to $\sqrt{\lambda_d}\boldsymbol{v}_d^i$.

$$\boldsymbol{Y}_i^{\mathrm{nt}} = \left[\sqrt{\lambda_1}\boldsymbol{v}_1^i, \cdots, \sqrt{\lambda_d}\boldsymbol{v}_d^i, \cdots, \sqrt{\lambda_m}\boldsymbol{v}_m^i\right]^{\mathrm{T}}$$

The parameters $\boldsymbol{Y}^{\mathrm{hp}}$ for happy face images in the training set are obtained similarly.

Obtain Low-dimensional Facial Parameters for an Input Neutral Face Image

By the property of manifold, each neutral face image and its K-nearest neighbors lie on or are close to a locally linear patch of the manifold. Therefore we could reconstruct an input neutral face image $\boldsymbol{I}_{\mathrm{un}}^{\mathrm{nt}}$ from its neighbors $\boldsymbol{I}_{(j)}^{\mathrm{nt}}$. Reconstruction errors are measured by the cost function as follows:

$$\varepsilon_I = \left\| \boldsymbol{I}_{\mathrm{un}}^{\mathrm{nt}} - \sum_{j=1}^{k} w_j \boldsymbol{I}_{(j)}^{\mathrm{nt}} \right\|^2$$

where the weights w_j summarize the contribution of the jth neutral face image in the training set to the input one's reconstruction. To compute the weights w_j, we minimize the cost function subject to the following constraint: $\sum_{j=1}^{k} w_j = 1$. The optimal weights w_j subject to this constraint are found by the following three steps.

Step 1 Evaluate inner products between neighbors to compute the neighborhood correlation matrix $\boldsymbol{C}_{ij} = \boldsymbol{I}_{(i)}^{\mathrm{nt}} \cdot \boldsymbol{I}_{(j)}^{\mathrm{nt}}$, and its matrix inverse $\boldsymbol{C}_{ij}^{-1}$;

Step 2 Compute the Lagrange multiplier $\lambda = \alpha/\beta$, which enforces the sum-to-one constraint, where

$$\alpha = 1 - \sum_{i=1}^{k}\sum_{j=1}^{k} \boldsymbol{C}_{ij}^{-1}(\boldsymbol{I}_{\mathrm{un}}^{\mathrm{nt}} \cdot \boldsymbol{I}_{(j)}^{\mathrm{nt}}),\ \beta = \sum_{i=1}^{k}\sum_{j=1}^{k} \boldsymbol{C}_{ij}^{-1}$$

Step 3 Compute the reconstruction weights:

$$w_j = \sum_{i=1}^{k} \boldsymbol{C}_{ji}^{-1}(\boldsymbol{I}_{\mathrm{un}}^{\mathrm{nt}} \cdot \boldsymbol{I}_{(j)}^{\mathrm{nt}} + \boldsymbol{\lambda}).$$

Inspired by the success of Locally Learning Embedding (LLE)[5], the characterization of local geometry in the image space is equally valid for local patches on the manifold. In particular, the same weights w_j that reconstruct the input neutral face image in the 96×128 dimensions should also reconstruct its embedded manifold coordinates in m dimensions. Therefore, we choose m-dimensional coordinates $\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{nt}}$ to minimize the embedding cost function:

$$\varepsilon_Y = \left\| \boldsymbol{Y}_{\mathrm{un}}^{\mathrm{nt}} - \sum_{j=1}^{k} w_j \boldsymbol{Y}_{(j)}^{\mathrm{nt}} \right\|^2$$

where w_j is obtained by the previous three steps. To obtain the optimal $\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{nt}}$, we simply set $\varepsilon_Y=0$, and obtain $\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{nt}}$ as: $\hat{\boldsymbol{Y}}_{\mathrm{un}}^{\mathrm{nt}} = \sum_{j=1}^{k} w_j \boldsymbol{Y}_{(j)}^{\mathrm{nt}}$.

Learn the Relationship between Neutral Facial Parameters and Happy Ones-in Pairs

The facial expression parameters $\boldsymbol{Y}_i^{\mathrm{nt}}$ and $\boldsymbol{Y}_i^{\mathrm{hp}}$ for the neutral and the happy face images respectively we obtained in the previous section lie on a Euclidean subspace. In order to infer the happy face image from the neutral one, we need to learn the relationship between $\boldsymbol{Y}_i^{\mathrm{nt}}$ and $\boldsymbol{Y}_i^{\mathrm{hp}}$ (see Fig. 5. 2). In this section, we use multiple linear regression to learn this relationship:

$$\boldsymbol{Y}^{\mathrm{hp}} = \boldsymbol{Y}^{\mathrm{nt}}\boldsymbol{\gamma} + \boldsymbol{\varepsilon} \tag{5-1}$$

where $\boldsymbol{Y}^{\mathrm{hp}}$ is an $N \times m$ matrix, N is the number of face images in the training set, and m is the number of facial expression parameters for happy faces; $\boldsymbol{Y}^{\mathrm{nt}}$ is also an $N \times m$ matrix; $\boldsymbol{\gamma}$ is an $m \times m$ matrix of parameters, which describes the relationship between pairs of face images that belong to the same person but have different facial expressions (neutral and happy); $\boldsymbol{\varepsilon}$ is an $N \times m$ matrix of random disturbances, and each of these errors has independent normal distribution with a zero mean and a constant variance.

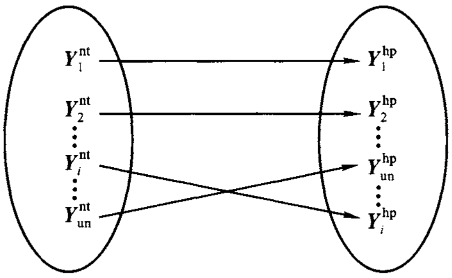


Fig. 5. 2 The relationship between neutral facial parameters and happy ones, and we learn the relationship by multivariate linear regression

Face Image Reconstruction from Low-dimensional Facial Parameters

We reconstruct face image from low-dimensional facial parameters by inverse mapping: $f^{-1}: Y \rightarrow I$, which is learnt by supervised learning techniques. For the case of N examples, input dimensionality is m and output dimensionality is D (e. g. 96×128). The approximation of the reconstructed happy face image is:

$$\hat{\boldsymbol{I}}_{\mathrm{un}}^{\mathrm{hp}} = \sum_{i=1}^{N} \boldsymbol{c}_i r(\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{hp}} - \boldsymbol{u}_i) \tag{5-2}$$

where $\boldsymbol{c}_i$ are vectors of weighting coefficients, r is the radial basis function, $\boldsymbol{u}_i$ are m-dimensional centers of the basis function which are defined as: $\boldsymbol{U}_i = \boldsymbol{Y}_i^{\mathrm{hp}}$.

Equation (5-2) can be described as $\hat{\boldsymbol{I}}_{\mathrm{un}}^{\mathrm{hp}} = \boldsymbol{C}\boldsymbol{g}(\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{hp}})$, where $\boldsymbol{C} = [c_1, \cdots, c_i, \cdots, c_N]$ is the vector of weights and $\boldsymbol{g}$ is the vector with the elements $\boldsymbol{g}_i = r(\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{hp}} - \boldsymbol{Y}_i^{\mathrm{hp}})$.

Matrix $\boldsymbol{G}$ can be defined as $\boldsymbol{G}_{ij} = r(\boldsymbol{Y}_i^{\mathrm{hp}} - \boldsymbol{Y}_j^{\mathrm{hp}})$ where weights $\boldsymbol{c}_i$ are obtained via solving the function $\boldsymbol{c}_i\boldsymbol{G} = \boldsymbol{I}_i^{\mathrm{hp}}$. The set of weights is given by $\boldsymbol{C} = \boldsymbol{I}^{\mathrm{hp}}\boldsymbol{G}^{-}$ where $\boldsymbol{I}^{\mathrm{hp}} = [\boldsymbol{I}_1^{\mathrm{hp}}, \cdots, \boldsymbol{I}_i^{\mathrm{hp}}, \cdots, \boldsymbol{I}_N^{\mathrm{hp}}]$ and $\boldsymbol{G}^{-}$ is the pseudo-inverse of $\boldsymbol{G}$. The happy face image $\boldsymbol{I}_{\mathrm{un}}^{\mathrm{hp}}$ is approximated by the linear combination of the examples $\boldsymbol{I}_i^{\mathrm{hp}}$:

$$\boldsymbol{I}_{\mathrm{un}}^{\mathrm{hp}} = \boldsymbol{I}^{\mathrm{hp}}\boldsymbol{G}^{-}\boldsymbol{g}(\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{hp}}) = \sum_{i=1}^{N} \boldsymbol{b}_i(\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{hp}})\boldsymbol{I}_i^{\mathrm{hp}}$$

where $\boldsymbol{b}(\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{hp}}) = \boldsymbol{G}^{-}\boldsymbol{g}(\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{hp}})$ and $\boldsymbol{b}_i$ is the column i of $\boldsymbol{b}$.

Inference

Given a face image $\boldsymbol{I}_{\mathrm{un}}^{\mathrm{nt}}$ with neutral facial expression, we obtain the low-dimensional parameters $\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{nt}}$. Based on equation (5-1), we are able to obtain the happy parameters $\boldsymbol{Y}_{\mathrm{un}}^{\mathrm{hp}}$. Afterwards, we reconstruct the happy face image $\hat{\boldsymbol{I}}_{\mathrm{un}}^{\mathrm{hp}}$ by inverse mapping f^{-1}. Several synthesized face images are shown in Fig. 5. 7(b). From these images, we can see that the synthesized happy face images look somewhat smooth, which will be improved by the local model in the next section.

5.1.1.2 Refinement by Local Patch-based Model

In this section, the residual face image R is generated from the original face image $\boldsymbol{I}$ minus the reconstructed face image $\hat{\boldsymbol{I}}$ (see Fig. 5. 5(a), (c) for samples of neutral and happy residual face image). Bayesian inference is then utilized to infer the plausible happy residual face image. Graph cuts based refinement is also taken into account.

We break the residual face image R into patches $\{PT_i\}$, and use a Markov network to probabilistically model the relationships between happy and neutral residual face image's patches, and between neighboring happy residual face image's patches. The width and height of each patch are denoted as $w(PT)$, $h(PT)$ respectively.

Markov Network

Markov network is used to model the spatial relationship between patches. In Fig. 5. 3, nodes are represented by circles, and the statistical relationship between nodes is described by edges. PT_i^{nt} is an observation, which denotes a patch

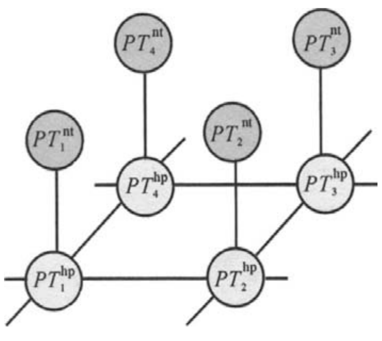


Fig. 5. 3 Pairwise Markov network

in the neutral residual face image R^{nt}; PT_i^{hp} is a hidden variable, which is a patch in the happy residual face image R^{hp}. PT_i^{hp} is what we seek to estimate. Given PT_i^{nt}, the conditional probability $p(\{PT_i^{\text{hp}}\}\mid\{PT_i^{\text{nt}}\})$ is:

$$p(\{PT_i^{\text{hp}}\} \mid \{PT_i^{\text{nt}}\}) = \frac{1}{Z}\prod_{(i,j)}\Psi_{ij}(PT_i^{\text{hp}}, PT_j^{\text{hp}})\prod_{i}\phi_i(PT_i^{\text{hp}}, PT_i^{\text{nt}}) \quad (5\text{-}3)$$

where Z is a normalization constant, Ψ_{ij} is the correlation function between the neighboring patches PT_i^{hp} and PT_j^{hp}, and ϕ_i is the compatible function between the patch PT_i^{nt} and the corresponding patch PT_i^{hp} in the happy residual face image, which is determined by robust optical flow.

Ψ_{ij} and ϕ_i

For Ψ_{ij}, we sample the neutral residual face image's nodes so that the happy residual face image's patches overlap with each other by $ol(PT)$ pixels. The sum of squared differences $D_{ij}(PT_i^{\text{hp}}, PT_j^{\text{hp}})$ in their overlapped region should be minimized. Ψ_{ij} is defined as:

$$\Psi_{ij}(PT_i^{\text{hp}}, PT_j^{\text{hp}}) = \exp\left(-\frac{D_{ij}(PT_i^{\text{hp}}, PT_j^{\text{hp}})}{2\sigma_r^2}\right)$$

where σ_r^2 is the Gaussian noise covariance.

For ϕ_i, we find the best matched patches in the training set for an input neutral patch $PT_{\text{in},i}^{\text{nt}}$. ϕ_i can be defined as:

$$\phi_i(PT_i^{\text{hp}}, PT_{\text{in},i}^{\text{nt}}) = \exp\left(-\frac{\| PT_i^{\text{nt}} - PT_{\text{in},i}^{\text{nt}} \|}{2\sigma_c^2}\right) \quad (5\text{-}4)$$

where σ_c^2 is also the Gaussian noise covariance, and $\| \cdot \|$ denotes Euclidean distance.

Fig. 5. 4 describes the computation of equation (5-4). For patches $PT_{\text{in},i}^{\text{nt}}$ input neutral face image, we find the best matched patches PT_i^{nt} of neutral face images in the training set. Then we use robust dense optical flow to find the corresponding patches PT_i^{hp} of happy residual face images in the training set. These patches are candidate scene patches for $PT_{\text{out},i}^{\text{hp}}$, which is the final hallucinated patch for the residual happy face image. Since optical flow is pixel-based, we down-sample the neutral residual face images and happy ones in the training set. Thus each patch in the original residual face images is translated to the corresponding pixel in the down-sampled images. Robust dense optical flow is then carried out on pairs of the down-sampled neutral/happy residual face images in the training set (see Fig. 5. 5). We set σ_c^2 to allow about six candidates within one standard deviation for each patch of the input neutral residual face image.

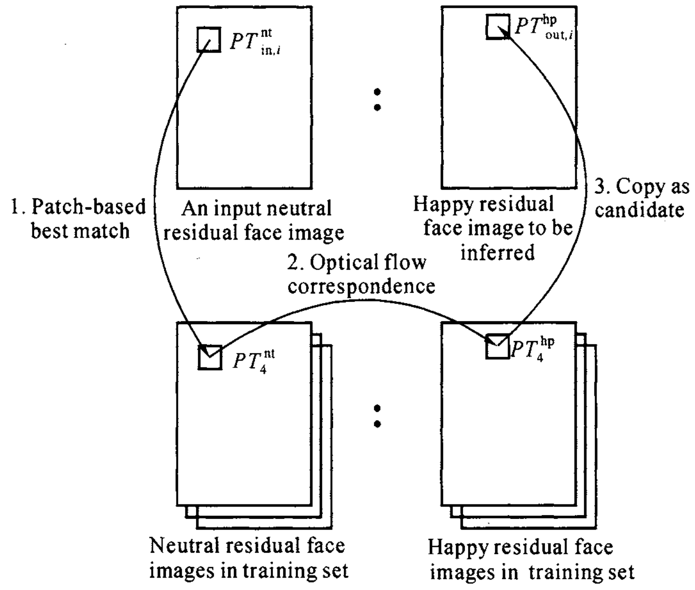

$PT^{nt}_{in,i}$
$PT^{hp}_{out,i}$
1. Patch-based best match
An input neutral residual face image
Happy residual face image to be inferred
3. Copy as candidate
2. Optical flow correspondence
PT^{nt}_{4}
PT^{hp}_{4}
Neutral residual face images in training set
Happy residual face images in training set

Fig. 5. 4 Diagram of belief propagation for happy residual face image inference

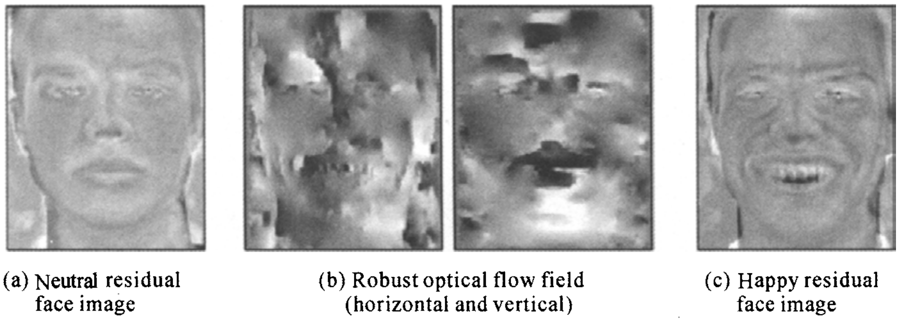

(a) Neutral residual face image
(b) Robust optical flow field (horizontal and vertical)
(c) Happy residual face image

Fig. 5. 5 Optical flow between the neutral and happy residual face images

Learn the Happy Residual Face Image by Belief Propagation

We need to find $PT^{hp}_{out,i}$ that maximizes $p(\{PT^{hp}_{out,i}\} \mid \{PT^{nt}_{in,i}\})$ in equation (5-3). Exact maximization of $p(\{PT^{hp}_{out,i}\} \mid \{PT^{nt}_{in,i}\})$ is NP hard, but belief propagation[6] has been successfully utilized in probabilistic graph model. In belief propagation, each node sends a message to all the neighboring nodes. The message from node i to node j is defined as:

$$m_{ij}(PT^{hp}_{out,j}) = \sum_{PT^{hp}_{out,i}} \Psi_{ij}(PT^{hp}_{out,i}, PT^{hp}_{out,j})\phi_i(PT^{hp}_{out,i}, PT^{nt}_{in,i}) \prod_{k \in NB(i)/j} m_{ki}(PT^{hp}_{out,i})$$

where $NB(i)$ denotes the neighboring nodes of node i.

The sum is over all patch candidates $PT^{hp}_{out,i}$ at node i. The product is over all neighbors of the sending node i except the receiving node j. After several iterations, the Markov network gets converged, and then the marginal probability B_i for each happy residual face image's patch $PT^{hp}_{out,i}$ at each node i is:

$$B_i(PT^{hp}_{out,i}) = c\phi_i(PT^{hp}_{out,i}, PT^{nt}_{in,i}) \prod_{j \in NB(i)} m_{ji}(PT^{hp}_{out,i})$$

where c is a normalization constant.

Then the optimal $\overline{PT^{hp}_{out,i}}$ at node i is $\overline{PT^{hp}_{out,i}} = \arg\max_{PT^{hp}_{out,i}} B_i(PT^{hp}_{out,i})$.

We can integrate the optimal residue patches together and form the optimal residue face image. The optimal residual face image plus the results of manifold inference is the final hallucinated happy face image, as is shown in Fig. 5. 7(c).

Refinement by Graph Cuts

The graph cuts algorithm is not conflicting with belief propagation. The output of belief propagation, i. e. , the happy residual face image is patch-based, and patches overlap with each other. The graph cuts algorithm can be utilized to find the best beam for the neighboring patches (see Fig. 5. 6). We utilize the source code provided by Boykov, et al. [7] to find the optimal beam for neighboring patches.

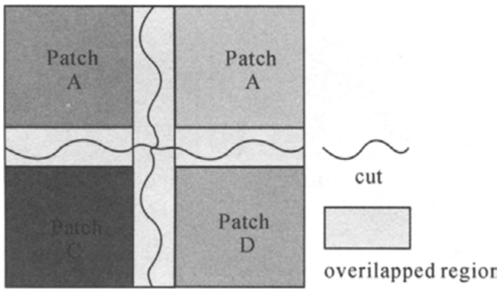


Fig. 5. 6 The overlapped region is optimally separated by graph cuts algorithm

5. 1. 1. 3 Experimental Results

We have built a prototype system to implement the algorithms in this section by Visual C++ and Matlab. Neutral and happy face images in the AR [8] face database are used as the source of our test face images. 300 face images of 150 different persons are selected as the training set, and other 30 images as the test data. We first need to crop the original face images to a size of 96 × 128 pixels. Then we mark five points in every face: center of eyes, top of nose, and corners of mouth, and then use affine transform to adjust the face images which ensure a rough correspondence among facial features of different faces. All the test face images are not included in the training set.

In global manifold learning/inference, we use ISOMAP to nonlinearly reduce the dimensionality to 18, accounting for 96% of the total variance. In local patch-based Markov network learning/inference, we choose $w(PT) = h(PT) = 5$ as the patch size and $ol(PT) = 2$ as the overlapping size in our experiments.

Fig. 5. 7 shows the experimental results. Using manifold learning and inference, we reconstruct the happy face images for input neutral ones. From Fig. 5. 7(b), it can be seen that manifold learning's reconstruction results are a little smooth. Then in the second step, Markov network based local low-level learning is carried out to make the happy face images more expressive (see Fig. 5. 7(c)).

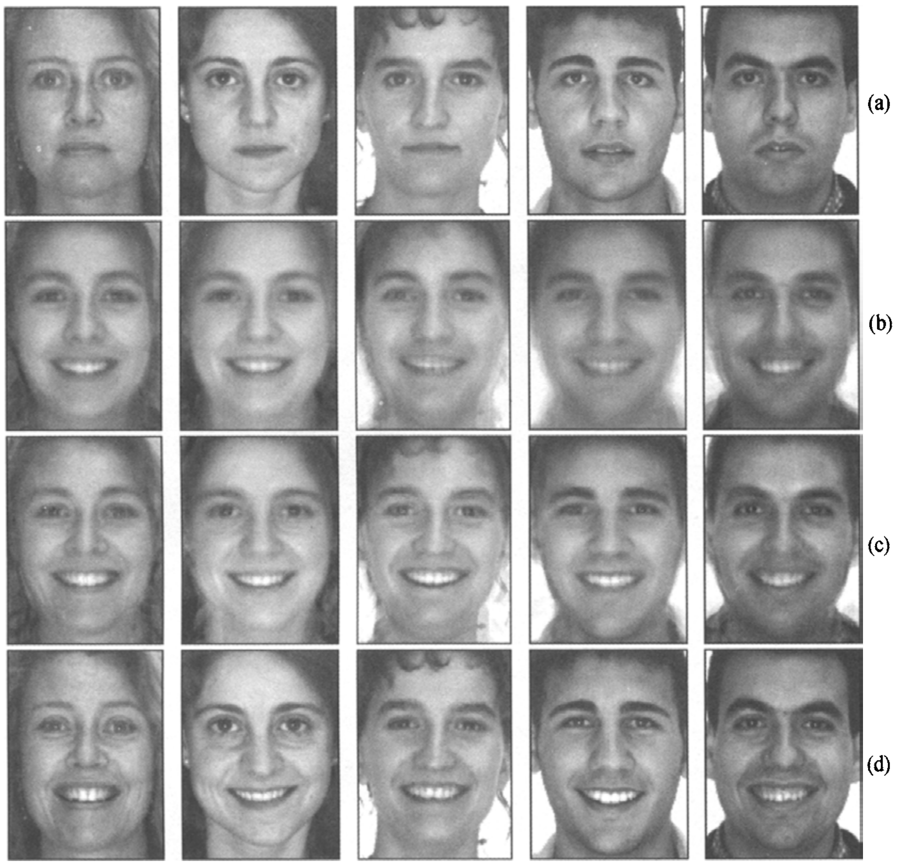


Fig. 5. 7 The results of hallucinated happy facial expressions. (a) The input neutral face images; (b) The preliminary results of manifold learning and inference; (c) The final results enhanced by belief propagation and graph cuts; (d) The ground-truth face images with happy facial expression

For a new input image with neutral expression, the person's teeth cannot be seen. Our algorithm can render his/her teeth on expressive face image, as shown in Fig. 5. 8. The teeth on these images look different from person to person, and the differences come from two reasons. One is manifold-based reconstruction, and the other is local Markov network inference. For manifold learning, each face image can be thought as one point in a high-dimensional space. Actually, there are two nonlinear manifolds: one is for neutral face images, and the other is for happy ones. A neutral face image is a point lying on the neutral manifold, and this also holds for a happy one. Since the neutral and happy face images are in pairs, the two corresponding points are in pairs too. We can seek the relationships among the pairs. If given two different neutral points, you will get two different happy points. Therefore, even for unseen facial features, e. g. teeth, we can generate different results for different input neutral face images. For Markov network learning, it further enhances the low-level features in the happy face images.

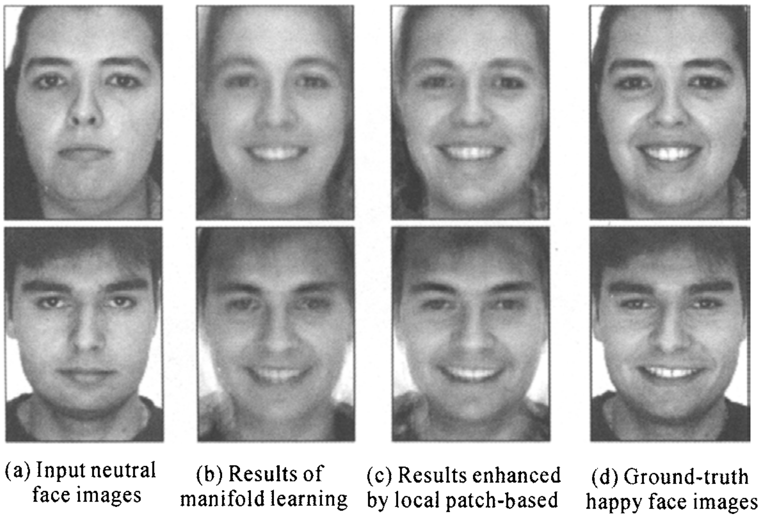

(a) Input neutral face images
(b) Results of manifold learning
(c) Results enhanced by local patch-based
(d) Ground-truth happy face images

Fig. 5. 8 For different person's unseen facial parts (e. g. teeth), the hallucinated happy face images are different too

The runtime of computing $\hat{I}_{\mathrm{un}}^{\mathrm{hp}}$ by manifold learning/inference is about 3 seconds, and the refinement by belief propagation and graph cuts takes about 2 seconds on a PentiumIV 1. 8 GHz CPU with 256 MB RAM.

In this section, we introduce a two-step facial expression hallucination approach based on sample learning. First, nonlinear manifold learning is used to estimate the preliminary smooth facial expression image, then the local details are refined by patch-based belief propagation algorithm. Currently we only show the results for gray-scale images, and it is easy to extend our algorithm for color images.

5.1.2 Video-based Facial Expression Hallucination

According to the above discussion, facial expression hallucination is an important approach to synthesizing facial expression. Existing hallucination approaches are image-based, i.e., synthesize specific facial expression image given an input face image with neutral facial expression. In this section, we propose a novel two-level video-based facial expression hallucination approach. Compared with existing works, the novelty mainly lies on two aspects: (1) We synthesize a dynamic expression video sequence instead of a static face image; (2) The approach is a fusion of local linear subspace learning and global nonlinear subspace learning, and provides a reasonable way to organize and represent the complex video sample space.

According to the two-level fusion framework, the local subspace learning adopts eigen-representation technique to compress video sequences in temporal domain, while the global subspace learning synthesizes the optimized facial expression in spatial domain. We will discuss the proposed video-based facial expression hallucination approach in detail in the following part.

5.1.2.1 Face Interest Region

Not all facial organs play important role in facial expression, and the eyes and mouth are more attractive in face to face communications. Thus in computer aided facial expression synthesis, we pay more attention to the regions near eyes and mouth, which are named as "face interest region" and can be described as the two rectangles in Fig. 5.9.

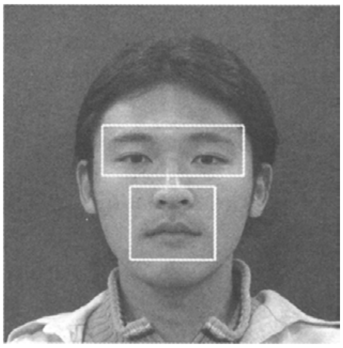


Fig. 5.9 Face interest region

Human face is a complex non-rigid surface, and it is difficult to model the whole face during facial expression, so we just focus on the face interest region. Thus a preprocess is needed to segment face interest regions frame by frame from the training video sequences. Segmentation is a classical and popular topic in digital image processing, and existing approaches mainly include: segmentation based on the gray-scale value of image pixels, segmentation based on the edge detection, and region growth algorithm. Different algorithms have different characteristics, but no algorithm is effective under arbitrary conditions. Furthermore, the segmentation results by the algorithms cannot be used as samples directly without normalization. So we employ simpler manual work to segment the interest region. First, find the position of pupils in an image, then move to four directions (up, down, left, right) by proper distances from the pupils, and we get the rectangular region around the eyes, this is the eye interest region. Second, we can determine the mouth area according to the position of pupils based on the topology of human facial organs. This is done by first determining the center of mouth interest region. The segmented human face interest regions are shown in Fig. 5. 10.

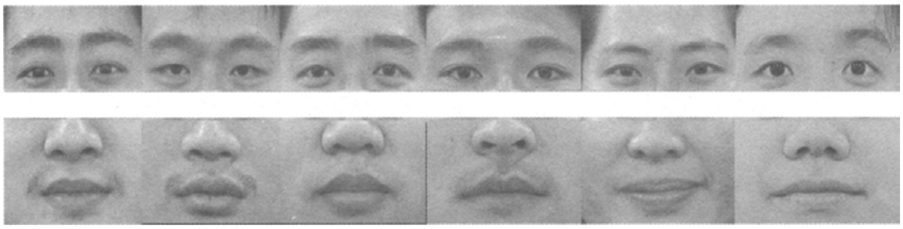


Fig. 5. 10 Face interest regions of sample faces

5. 1. 2. 2 Two-level Hierarchical Fusion Framework

The training samples comprise tens of video clips, and each video clip represents one kind of facial expression of a specific person from neutral to apex. Our goal is to hallucinate expression video sequences of a new test subject based on sample videos, and the input is a single frontal face image with neutral expression.

Due to the high-dimensionality of video samples, organizing the sample space well becomes a challenging problem. Here, we propose a hierarchical approach to perform training and synthesizing, which includes two levels: local linear subspace learning and global nonlinear subspace learning.

Local Linear Subspace Learning

In this level, each training sample (a video sequence) is considered to construct a local linear subspace. Principal Component Analysis (PCA) has been proved effective in learning such a linear subspace. So, in this level, we use PCA to compute eigen-representation of each sample offline for further use.

Step 1 Given a video sequence, we stack columns of each frame into one vector and integrate all the vectors to form a sample matrix $\boldsymbol{X}$.

Step 2 Compute $\tilde{\boldsymbol{X}}=(\boldsymbol{X}-\bar{\boldsymbol{X}})/\sqrt{N}$ to register $\boldsymbol{X}$, where $\bar{\boldsymbol{X}}$ is the mean value and N is the number of samples. To deal with the problem caused by high-dimensionality, we perform QR factorization to gain $[\boldsymbol{q},\boldsymbol{r}]=QR(\tilde{\boldsymbol{X}})$, then Singular Value Decomposition (SVD) is imposed on $\boldsymbol{r}$ to get $[\boldsymbol{u},\boldsymbol{s},\boldsymbol{v}]=SVD(\boldsymbol{r})$, and then eigenvectors can be obtained by $\boldsymbol{U}=\boldsymbol{q}\cdot\boldsymbol{u}$, we do so for solving the problem in a numerically stable way.

Step 3 Thus, we can project any frame $\boldsymbol{f}$ on $\boldsymbol{U}$ to get the reconstruction coefficients $\boldsymbol{y}=\boldsymbol{U}^{\mathrm{T}}\cdot(\boldsymbol{f}-\bar{\boldsymbol{X}})$, and $\boldsymbol{f}$ can be reconstructed by $\tilde{\boldsymbol{f}}=$

$U \cdot y + \bar{X}$. So, we store each sample's eigenvectors U, coefficients y and mean value $\bar{X}$ as the eigen-representation for expression synthesis.

Global Nonlinear Subspace Learning

In global nonlinear subspace learning, given an input face image with neutral expression as a high-dimensional data point, we aim at finding its nearest k neighbors among the first frames (neutral) of the video samples by Locally Learning Embedding (LLE) algorithm[9]. LLE is an unsupervised manifold learning algorithm that computes low-dimensional, neighborhood preserving embeddings of high-dimensional input and recovers the global nonlinear structure from locally linear fits. According to LLE, each high-dimensional data point can be reconstructed by weighted linear combination of its neighbors. The reconstruction weights reflect intrinsic geometric properties of the data that are invariant when high-dimensional data points are transformed to low-dimensional coordinates. The process of LLE algorithm can be briefly described as three steps listed below.

Step 1 Find k neighbors for each sample data. Selecting K-closest neighbors for each point using a distance measure such as the Euclidean distance. Solving for the manifold reconstruction weights. The reconstruction errors are measured by the cost function:

$$\varepsilon(w) = \sum_{i=1}^{N} \| \boldsymbol{X}_i - \sum_{j=1}^{N} \boldsymbol{W}_{ij} \boldsymbol{X}_{ij} \|^2$$

where $\boldsymbol{X}_i$ is a data point and $\varepsilon(\boldsymbol{w})$ is the sum of the squared distances between all data points and their reconstruction neighbors. The weights $\boldsymbol{W}_{ij}$ represent the contribution of the jth data onto the ith reconstruction. Two constraints should be obeyed: $\sum \boldsymbol{W}_{ij} = \boldsymbol{1}$, $\boldsymbol{W}_{ij} = \boldsymbol{0}$ if $\boldsymbol{X}_j$ is not a neighbor of $\boldsymbol{X}_i$. The weights are then determined by a least squares minimization of the reconstruction errors. To solve the weights, we need to construct a local covariance matrix $\boldsymbol{Q}^i$ as $\boldsymbol{Q}^i_{im} = (\boldsymbol{X}_i - \boldsymbol{X}_{ij})^{\mathrm{T}} (\boldsymbol{X}_i - \boldsymbol{X}_{im})$. Combine this equation with $\sum \boldsymbol{W}_{ij} = \boldsymbol{1}$, the local optimal reconstruction weight matrix can be obtained as:

$$\boldsymbol{w}^i_j = \sum_{m=1}^{k} (\boldsymbol{Q}^i)^{-1}_{jm} \Big/ \sum_{p=1}^{k} \sum_{q=1}^{k} (\boldsymbol{Q}^i)^{-1}_{pq} .$$

In practical computation, $\boldsymbol{Q}^i$ may be a singular matrix, thus a normalization process should be adopted to ensure the positive definite property which can be described as $\boldsymbol{Q}^i = \boldsymbol{Q}^i + \boldsymbol{rI}$, where $\boldsymbol{r}$ is the normalization parameter and $\boldsymbol{I}$ is a $K \times K$ unitary matrix.

Step 2 Mapping each high-dimensional data to a low-dimensional coordinate. This is done by minimizing the cost function representing locally linear reconstruction errors:

$$\Phi(\boldsymbol{Y}) = \sum_{i=1}^{N} \| \boldsymbol{Y}_i - \sum_{j=1}^{N} \boldsymbol{W}_{ij} \boldsymbol{Y}_{ij} \|^2$$

where $\Phi(\boldsymbol{Y})$ is the reconstruction error, $\boldsymbol{Y}_i$ is the low-dimensional coordinate corresponding to $\boldsymbol{X}_i$, $\boldsymbol{Y}_{ij}$ is K-nearest neighbors of $\boldsymbol{Y}_i$,

$$\sum_{i=1}^{N} \boldsymbol{Y} = \boldsymbol{0},\ \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{Y}_i \boldsymbol{Y}_i^{\mathrm{T}} = \boldsymbol{I}$$

where $\boldsymbol{I}$ is an $m \times m$ unitary matrix. $w_j^i (i=1,\cdots,N)$ can be stored in an $N \times N$ sparse matrix $\boldsymbol{W}$, when $\boldsymbol{X}_{ij}$ is a neighbor of $\boldsymbol{X}_i$, $\boldsymbol{W}_{ij} = w_j^i$, else $\boldsymbol{W}_{ij} = \boldsymbol{0}$. We rewrite the error function as:

$$\min \varepsilon(\boldsymbol{Y}) = \sum_{i=1}^{N} \sum_{j=1}^{N} \boldsymbol{M}_{ij} \boldsymbol{Y}_i^{\mathrm{T}} \boldsymbol{Y}_j$$

where $\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{W})^{\mathrm{T}} (\boldsymbol{I} - \boldsymbol{W})$ is a symmetry matrix. To minimize the error function, $\boldsymbol{Y}$ is selected as m eigenvectors corresponding to the smallest m non-zero eigenvalues of $\boldsymbol{M}$. Actually, the first eigenvalue is almost zero, so we omit it and choose the eigenvectors corresponding to the eigenvalues with the order from 2 to $m+1$.

After LLE implementation, we gain the low-dimensional coordinates of both the neighbor samples and the input image, then expression sequence synthesis can be performed based on the low-dimensional coordinates and corresponding eigen-representations, which have been computed through local linear subspace learning. Indeed, the two-level hierarchical approach is a fusion of nonlinear and linear subspace learning, where the local level aims at simplifying the video hallucination by eigen-representation in temporal domain, while the global level contributes to providing the optimized expression appearance in spatial domain. The detailed expression synthesis procedure will be discussed in the next section.

5.1.2.3 Video-based Dynamic Facial Expression Hallucination

Our proposed hierarchical fusion approach performs the local linear learning offline only once, while the global nonlinear learning is performed each time when a test subject comes. The proposed approach includes three steps.

Step 1 Let $\boldsymbol{I}_{\text{in}}$ be the input subject image and $\boldsymbol{I}_{\text{tr}}$ be the first frames of N training samples, we integrate $\boldsymbol{I}_{\text{in}}$ and $\boldsymbol{I}_{\text{tr}}$ into one matrix form (each image can be stacked into one column vector and the matrix is composed of $N+1$ vectors).

Step 2 Find the N-nearest neighbors of $\boldsymbol{I}_{\text{in}}$ in $\boldsymbol{I}_{\text{tr}}$ by LLE, also, the reconstruction weights as well as the low-dimensional manifold coordinates $\boldsymbol{Y}_{\text{in}}$ and $\boldsymbol{Y}_{\text{tr}}$ of these images are simultaneously computed, here $\boldsymbol{Y}_{\text{in}}$ corresponds to $\boldsymbol{I}_{\text{in}}$ and $\boldsymbol{Y}_{\text{tr}}$ corresponds to $\boldsymbol{I}_{\text{tr}}$. The low-dimensional coordinates of $\boldsymbol{Y}_{\text{in}}$'s N-nearest neighbors can be denoted as $\boldsymbol{Y}_{\text{nb}}$.

Step 3 Since the eigenvectors $\boldsymbol{U}$, coefficients $\boldsymbol{y}$ and mean value $\bar{\boldsymbol{X}}$ of each video sample have been computed through local level learning, we choose $\boldsymbol{U}_{\text{nb}}$, $\boldsymbol{y}_{\text{nb}}$, $\bar{\boldsymbol{X}}_{\text{nb}}$ (the eigen-representations) of those nearest neighbors $\boldsymbol{y}_{\text{nb}}$ and as training data to synthesize the eigenvectors $\boldsymbol{U}_{\text{in}}$, coefficients $\boldsymbol{y}_{\text{in}}$ and mean value $\bar{\boldsymbol{X}}_{\text{in}}$ of the expression sequence corresponding to the input image $\boldsymbol{I}_{\text{in}}$. According to PCA rationale, the eigen-representation is adequate to represent a local linear subspace. Thus, arbitrary frame $\tilde{\boldsymbol{f}}_{\text{in}}$ in the video sequence corresponding to image $\boldsymbol{I}_{\text{in}}$ can be reconstructed as $\tilde{\boldsymbol{f}}_{\text{in}}=\boldsymbol{U}_{\text{in}} \cdot \boldsymbol{y}_{\text{in}}+\bar{\boldsymbol{X}}_{\text{in}}$ according to PCA theory.

In Step 3, the correspondence between the eigen-representations and the low-dimensional coordinates are simulated by a Radial Basis Function (RBF) regression. The RBF regression function takes the form:

$$r_t = \beta_0 + \sum_{i=1}^{k} \beta_i K(\boldsymbol{x}_t, \boldsymbol{\mu}_i)$$

where $\boldsymbol{x}_t \in \mathbf{R}^d$ and $r_t \in \mathbf{R}$ are input training data, $\boldsymbol{\beta}=(\beta_0, \cdots, \beta_k) \in \mathbf{R}^{k+1}$ is a vector of regression coefficients. K is a local kernel function centered on $\boldsymbol{\mu} \in \mathbf{R}^d$. In order to simplify the regression problem, we first perform K-NN clustering algorithm on input training data $\boldsymbol{x}_t$ and assign the kernel function centers $\boldsymbol{\mu}$ to be the clustering centers. Suppose r_t, $\boldsymbol{x}_t$ as well as the kernel function K are available, the regression parameter $\boldsymbol{\beta}=(\beta_0, \cdots, \beta_k) \in \mathbf{R}^{k+1}$ can be solved by a standard least squares algorithm.

In our implementation, $\boldsymbol{x}_t$ is the neighbors' low-dimensional coordinates $\boldsymbol{Y}_{\text{nb}}$, and $\boldsymbol{r}_t$ is the neighbors' eigen-representation which takes the form: $\boldsymbol{r}_t = (\boldsymbol{U}_{\text{nb}}, \boldsymbol{y}_{\text{nb}}, \bar{\boldsymbol{X}}_{\text{nb}})$. After $\boldsymbol{\beta}$ is calculated, given $\boldsymbol{Y}_{\text{in}}$ as input, the eigenvectors $\boldsymbol{U}_{\text{in}}$, coefficients $\boldsymbol{y}_{\text{in}}$ and mean value $\bar{\boldsymbol{X}}_{\text{in}}$ of the expression sequence corresponding to the input image are synthesized. So according to PCA theory, the new expression sequence corresponding to the input neutral face can be reconstructed frame by frame through:

$$\boldsymbol{f}=\boldsymbol{U}_{\text{in}} \cdot \boldsymbol{y}_{\text{in}}+\bar{\boldsymbol{X}}_{\text{in}}$$

5.1.2.4 Facial Expression Video Database

Existing Face Database

There already exist some face databases for the purpose of scientific research. The representative databases include: FERET database, XM2VTS database, Yale database, AR database, MIT database, ORL database and PIE face database. Now we will introduce the databases in brief.

FERET database: most widely used face database, created by FERET project, comprises of 14,051 face images with multiple poses and illumination conditions. Most people in the database are Caucasian.

XM2VTS database: includes images, audio and video sequences of 295 participants captured at different times. Every time for each participant, 2 video clips with different head rotation and 6 audio video sequences are captured. In addition, the 3D models of 293 people are available.

Yale database: constructed by the computer vision and control center of Yale University, and includes 165 images of 15 volunteers with illumination, expression and pose variations. The illumination and pose condition are strictly controlled so as to facilitate the modeling and analysis of illumination and pose change.

AR database: created by the computer vision center of Barcelona, and includes 3,288 images of 116 people. The camera parameters and illumination condition are under strict control.

MIT database: constructed by the media lab of MIT, includes 2,592 images from 16 volunteers with variant pose, illumination and scale.

ORL database: built by AT&T Lab of Cambridge, includes 400 face images of 40 people. The images of some volunteers were captured under different pose, expression and facial decoration. ORL was often used in early period of face recognition research, however, due to the small data quantity, ORL becomes unpractical in recent years.

PIE database: created by CMU, comprises 41,368 images of 68 volunteers with multiple pose, illumination and facial expression. The illumination and pose conditions are rigidly controlled.

Above discussion indicates that most face databases include face images with different pose, illumination and facial expression. However, the sample images are all in gray-scale manner and video samples rarely appear in these databases. Therefore, it is necessary to set up our own facial expression video database according to the requirement of application.

Facial Expression Video Database

To capture the facial expression video samples, we use a Sony HDV 1080i video camera recorder, and the video frame resolution amounts to 19,201,080. To ensure good performance, the actors are informed to perform multiple expressions from neutral to apex with no rigid movements of the head in a room with average illumination. The distance between human face and the camera is fixed at 2 meters with known camera inner parameters.

Our facial expression database includes 232 video sequences covering 4 kinds of typical expressions (happy, angry, surprise, fear) coming from 58 individuals, each video sequence is normalized to 20 video frames. Since the most sensitive parts of a human face are the eyes and the mouth, we separate the faces in 4,640 video frames (232 videos and 20 frames per video sequence) manually into eye and mouth interest regions and treat them respectively during training and synthesizing. The methods for hallucinating the eyes and the mouth are totally identical.

5.1.2.5 Results and Discussions

After the dynamical sequence of eyes and mouths are synthesized through the proposed approach, the synthesized face interest regions are transplanted onto the input image manually to form the dynamic facial expression sequence. When extracting the face interest regions of the sample videos in training process, we have labeled the positions of pupils manually, so we just need to mark out the pupils of input face, and align the input face image with the synthesized eye interest regions according to the positions of pupils. The mouth interest region can be treated in a similar way, while the mouth position of the input face image can be obtained according to the pupils' positions based on the facial organ topology. In this way, we can achieve reasonable fusion of input face image and synthesized face interest regions. To deal with the 24 bit true color video frames, our approach is applied on the *R*, *G*, *B* channels respectively, and the final results are the integration of the three channels. To testify the efficacy of our approach, an arbitrary face image with neutral expression is selected from the database as input face image, and other sample videos are used to synthesize expression video sequences with 20 frames each. The hallucination process takes no more than 10 seconds in a PentiumIV 2.4 GHz CPU with 1 G RAM. We perform the "cross validation" process (randomly pick one out of training data) 10 times and part of the experimental results are demonstrated in Figs. 5.11—5.14. In each figure, the first row is the ground truth facial expression sequence, and the second row is the hallucinated facial expression sequence.

Of course, besides RBF, there is other way to synthesize facial expres-

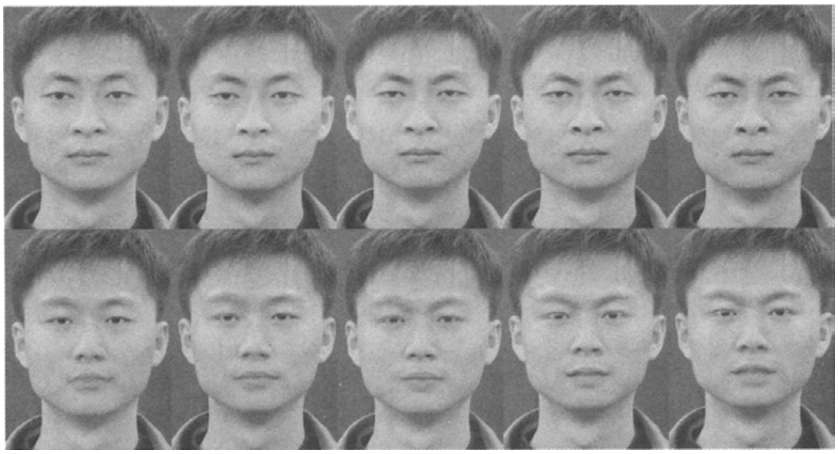


Fig. 5.11 The ground truth and the hallucinated angry expression

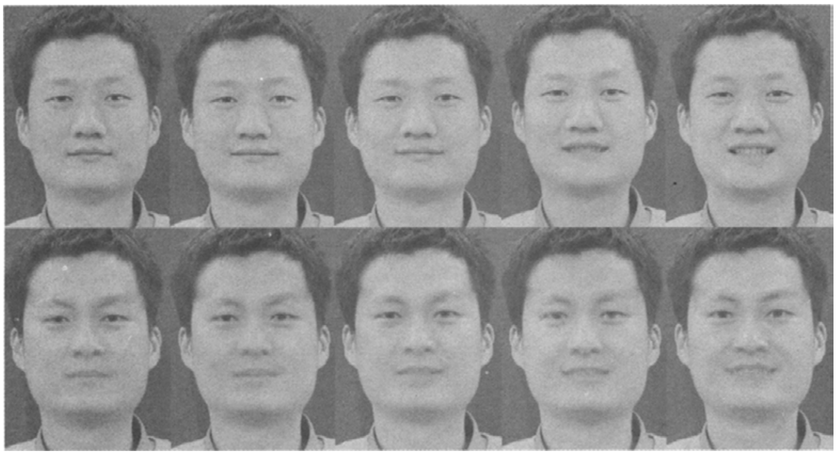


Fig. 5. 12 The ground truth and the hallucinated happy expression

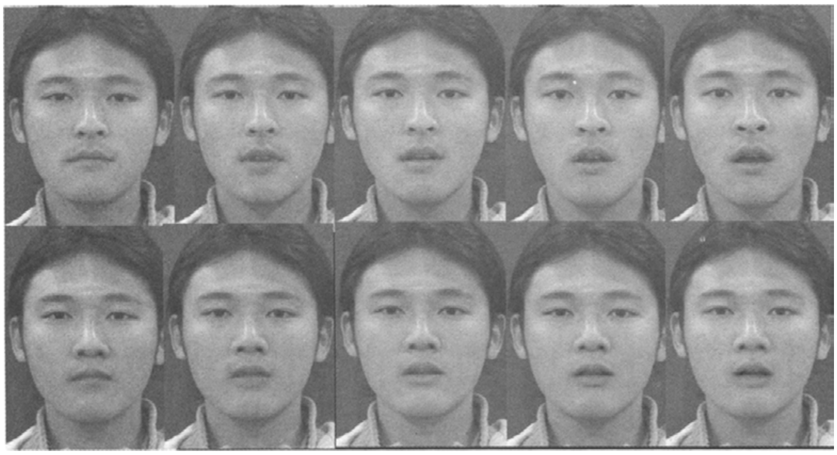


Fig. 5. 13 The ground truth and the hallucinated surprise expression

sion sequences given sample data. Remember that in Step 2 of section "Video-based Dynamic Facial Expression Hallucination", LLE algorithm obtains not only K-closest neighbors of input face image, but also the weights for the neighbors used to reconstruct the input image through linear combination. Compared with the linear combination approach, the RBF regression approach proposed in this section maintains more high frequency information, see Fig. 5. 15.

Through the global nonlinear learning, we have computed the low-dimensional coordinates of the input subject and the training samples. It is proved that two factors may influence the final hallucination results, i. e. the neighborhood size of LLE and the dimensionality of the low-dimensional coordinates. The average Root Mean Square (RMS) error of 10 cross

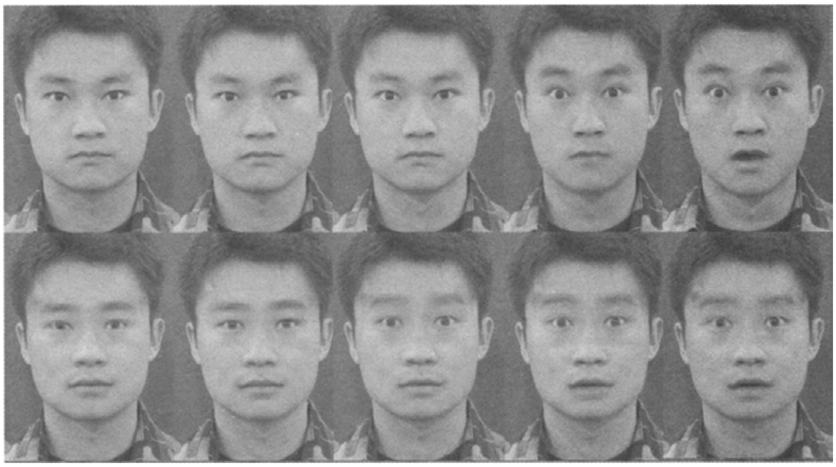
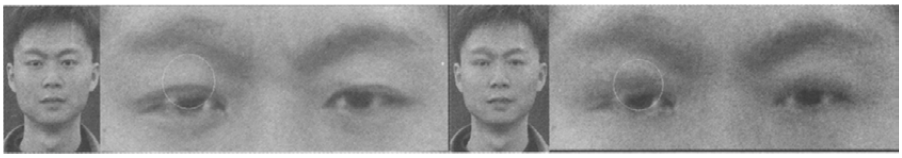


Fig. 5. 14 Also the ground truth and the hallucinated surprise expression



(a) (b)

Fig. 5. 15 (a) Hallucination result by RBF regression; (b) Hallucination result by linear combination. The right side of either (a) or (b) is the magnified eye region. Note that near the region of the circle, the result by weighted linear combination method looks smoother than the RBF regression approach

validation tests using different neighborhood size and dimensionality are shown in Figs. 5. 16 and Fig. 5. 17. The RMS error is computed according to the only input original image and the first frame of the hallucinated expression sequences. We compare the RMS error of our approach with that of the weighted linear combination method and verify the superiority of our approach.

Fig. 5. 16 indicates that the RMS error is very unstable when the neighborhood size is less than 8. We adjust the neighborhood size empirically, when the neighborhood size is between 8 and 17, the RMS error remains relatively stable at lower values, when the neighborhood size surpasses 17, the RMS error rises dramatically. Fig. 5. 17 shows that in global LLE learning, when the dimensionality of the low-dimensional coordinates is between 8 and 16, the mean RMS error remains at lower values, otherwise, the RMS error rises dramatically.

Though the neighborhood size and the dimensionality of the low-dimensional coordinates do influence the hallucination results, there lacks perfect approach to determine these parameters automatically. In many applica-

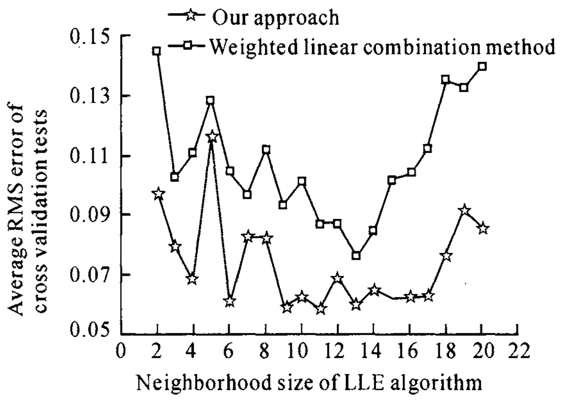


Fig. 5.16 The average RMS error under different neighborhood size

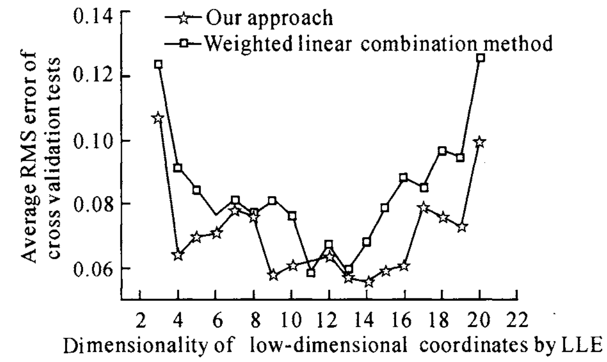


Fig. 5.17 The average RMS error under different dimensionality of the low-dimensional coordinates computed by LLE

tions, these parameters are determined empirically according to different cases. In our experiments, the neighborhood size and the dimensionality of the low-dimensional coordinates are fixed at 11 and 9, respectively.

In this section, we present a novel two-level hierarchical fusion approach to hallucinate facial expression sequences from training video samples given only one frontal face image with neutral expression. According to the fusion approach, the local linear subspace learning is combined with the global nonlinear subspace learning, the local level simplifies the video hallucination by eigen-representation of the samples in temporal domain, and the global level creates the optimized expression appearance in spatial domain. The two-level hierarchical fusion approach provides a sound solution to the problem of organizing the complex training video sample space, and this is the main contribution of our work. Our approach generates reasonable facial expression sequences with little artifact compared with existing methods.

5.2 Video-based Facial Expression Capture

Tracking for multiple facial features is a challenging and important topic in computer vision area. Kass, et al. proposed Snakes (Active Contour Model) to track the contour of lips. Snakes are energy-minimizing splines guided by constraints. They can be used to obtain smooth feature contours. But when the total number of control points is large (e. g. dozen), the dimensionality is too high, which will decrease the tracking efficiency. Furthermore, allowing arbitrary variation in positions of control points over time will lead to instability in tracking. Cootes, et al.[10] proposed the ASM/AAM algorithms, in which tracking is based on face detection and recognition. However the tracking results depend on the model's initial position and the variations contained in the training set, which makes it difficult to deal with occlusions. Furthermore, during the training, broken line is used to mark the facial features, which is not smooth. Current facial feature tracking approaches are mainly based on single feature. Multiple facial feature tracking faces the problems of both global rigid motion and local facial features' non-rigid motion, so multiple feature tracking is more difficult. The non-rigid motion of facial features is rapid, and the motion of multiple features will interfere with each other. Kalman filter based tracking is inadequate because it is based on Gaussian probability distribution. The probability density of facial feature is a mixture of several distributions rather than single Gaussian in case of interference.

Thus, in this section, we will discuss two kinds of approaches for multiple facial feature tracking.

5.2.1 Multiple Facial Feature Tracking Based on Bayesian Network Enhanced Prediction Model

The algorithm can be described as the following steps in brief.

Step 1 Choose the B-spline to describe the feature's contour. B-spline is smooth and better than broken line, since the contour of facial feature is also smooth.

Step 2 Utilize PCA to reduce dimensionality for each facial feature. Therefore unplausible contours are eliminated by subspace method.

Step 3 Propose a multi-cue based prediction model.

(1) First, use low-level feature based face tracking algorithm (Mean-shift) to give an estimation of the face's position. Therefore the search space for observation model is narrowed down.

(2) Multiple facial features are tracked simultaneously, and the spatial constraint among facial features is also taken into account.

(3) We learn the second-order Auto Regressive Process (ARP) based

dynamic model for each facial feature.

(4) We use graphical model — Bayesian network to enhance the ARP based dynamic model. The Bayesian network in this section combines the influence on a facial feature in the current time instant contributed by multiple facial features in the previous time instant. In this way, it is more robust than tracking each facial feature independently. We integrate all the above prediction models as multi-cues into the prediction model of Kalman filter.

Step 4 Finally, the prediction and observation model make up the Kalman filter framework in the standard way.

5.2.1.1 Representation of Facial Feature's Contour

We track the contours of facial features, and use B-spline ($x(u,t)$, $y(u,t)$) to represent facial feature's contour in time instant t. Suppose there are N spans and N_c control points, we have

$$\boldsymbol{x}(u,t)=\boldsymbol{B}(u)\boldsymbol{C}^x(t),\ \boldsymbol{y}(u,t)=\boldsymbol{B}(u)\boldsymbol{C}^y(t)\ (0\leqslant u\leqslant N) \tag{5-5}$$

where $\boldsymbol{C}^x=[C_1^x,\cdots,C_{N_c}^x]^{\mathrm{T}}$ are the x coordinates for all control points in time instant t, and $\boldsymbol{C}^y$ are the y coordinates. To closed curves, the number of control points equals that of the spans: $N_c=N$, while to open curves, $N_c=N+d$. Vector $\boldsymbol{B}$ is the blending parameter:

$$\boldsymbol{B}(u)=[B_1(u),B_2(u),\cdots,B_{N_c}(u)] \tag{5-6}$$

where $B_i(u)$ is the basis function of the B-spline. The control points of B-spline composite a spline vector $\boldsymbol{C}(t)=[\boldsymbol{C}^x(t),\boldsymbol{C}^y(t)]^{\mathrm{T}}$.

The B-splines that represent facial features are shown in Fig. 5. 18, where the contour of eyebrow is the upper edge of eyebrow; the contour of eye is the boundary between eyelid and eyeball, excluding the eyelid; the contour of nose is the border between nose and the skin of face; and the contour of mouth is the edges of upper and lower lips.

If we manipulate the control point's positions arbitrarily, it is easy to generate a spline that does not look like facial feature's contour (see Fig. 5. 19). Therefore arbitrarily manipulating the control points may lead to tracking failure.

5.2.1.2 Dimensionality Reduction for Contour of Facial Feature

The movement of facial feature can be decomposed into two parts: rigid motion and non-rigid motion. The rigid motion is caused by the motion of head, while the non-rigid one is the motion of each facial feature that caused by facial expressions. Facial features (eyes, eyebrows, nose, mouth) are generally in the same plane. When a rigid motion of head occurs, the contour of facial feature has six degrees of freedom (DOF). For

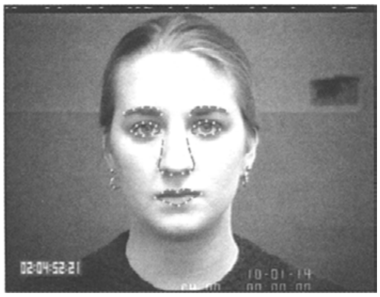


Fig. 5.18 The facial feature contour represented by B-spline

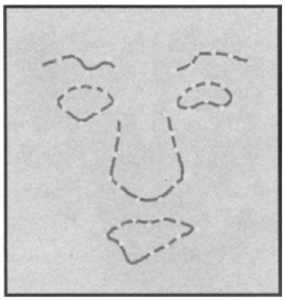


Fig. 5.19 The results of arbitrarily manipulating spline vectors

the non-rigid motion of a facial feature, we carry out PCA for the contour of facial feature in the training face image sequence. Suppose the dimensionality of non-rigid motion is reduced to N_{nr}, and then the total dimensionality of all facial features is $6 + N_{nr}$. Let $\boldsymbol{s}_t$ denote the parameters of state space after the dimensionality reduction, the spline vector $\boldsymbol{C}(t)$ can be written as:

$$\boldsymbol{C}(t)=\boldsymbol{W}\boldsymbol{s}_t+\boldsymbol{C}_0 \tag{5-7}$$

where $\boldsymbol{W}$ is a $N_c \times N_s$ shape matrix. N_c denotes the DOF before dimensionality reduction, and $N_c=2N_s$. N_s denotes the dimensionality after dimensionality reduction, and $N_s=6+N_{nr}$. $\boldsymbol{C}_0$ is the template of contour, which is usually obtained by manually marking. During tracking, it is not proper to allow arbitrary change of control points. When dealing with non-rigid motion, Snake tracking lacks robustness with too many degrees of freedom.

5.2.1.3 Multi-Cue Based Prediction Model

Second Order Auto Regressive Process Based Prediction Model

The motion of a facial feature's contour can be modeled by a noise driven second-order ARP, which can be shown as the following second-order linear differential equation:

$$\boldsymbol{s}_t=\boldsymbol{A}_2\boldsymbol{s}_{t-2}+\boldsymbol{A}_1\boldsymbol{s}_{t-1}+\boldsymbol{D}_0+\boldsymbol{B}_0\boldsymbol{w}_t \tag{5-8}$$

where $\boldsymbol{A}_1$, $\boldsymbol{A}_2$ and $\boldsymbol{B}_0$ are matrices, $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are the deterministic parameters, and $\boldsymbol{B}_0$ is the stochastic parameter. $\boldsymbol{D}_0$ denotes a fixed offset, and the distribution of each component of $\boldsymbol{w}_t$ belongs to i. i. d. Gaussian noise. Let $\boldsymbol{\chi}_t=[\boldsymbol{s}_{t-1},\boldsymbol{s}_t]^{\mathrm{T}}$, then equation (5-8) can be written as:

$$\boldsymbol{\chi}_t=\boldsymbol{A}\boldsymbol{\chi}_{t-1}+\boldsymbol{D}+\boldsymbol{B}\boldsymbol{w}_t \tag{5-9}$$

where

$$\boldsymbol{A}=\begin{bmatrix}\boldsymbol{0} & \boldsymbol{I}\\ \boldsymbol{A}_2 & \boldsymbol{A}_1\end{bmatrix},\ \boldsymbol{D}=\begin{bmatrix}\boldsymbol{0}\\ \boldsymbol{D}_0\end{bmatrix},\ \boldsymbol{B}=\begin{bmatrix}\boldsymbol{0}\\ \boldsymbol{B}_0\end{bmatrix},$$

and $\boldsymbol{I}$ is an identity matrix. From equation (5-9), we can see that $\boldsymbol{\chi}_t$ only depends on $\boldsymbol{\chi}_{t-1}$ in the previous time instant. Therefore the dynamic model for one facial feature is actually a Markov chain. But this model doesn't consider the relationship among facial features. Since second-order ARP can describe constant velocity motion, decay and damped oscillation, we use it as the plausible dynamic model for each facial feature.

In the second-order ARP dynamic model (see equation (5-9)), the parameters $\boldsymbol{A}$, $\boldsymbol{D}$ and $\boldsymbol{B}$ are unknown. Although it is possible to specify the parameters empirically, it is more convincible to estimate these parameters from training image sequences. In this section, we choose a bootstrapping strategy to learn the parameters for the dynamic model.

Firstly, we build the dynamic model according to the pre-assigned parameters empirically, and track the simple and slow facial motion feature, thus a sequence of motion feature $\boldsymbol{\chi}_1^1, \cdots, \boldsymbol{\chi}_M^1$ can be obtained in training process, where M is the number of frames in training video. Given $\boldsymbol{\chi}_i^1$, the solution to $\boldsymbol{A}$, $\boldsymbol{D}$ and $\boldsymbol{B}$ is a standard Expectation Maximization (EM) problem. Supposing the initial parameters gained by EM training are $\bar{\boldsymbol{A}}^1$, $\bar{\boldsymbol{D}}^1$ and $\bar{\boldsymbol{B}}^1$, a new dynamic model can be built using these parameters, and the refinement of tracking is done based on the new model. Generally speaking, 2 to 3 iterations are enough to generate an effective dynamic model.

According to the Markov property of the dynamic model, if the tracking results of two time points are known as $\boldsymbol{s}_{t-2}$ and $\boldsymbol{s}_{t-1}$, the state space of feature contour at current time point can be estimated as $\boldsymbol{s}_t$ according to equations (5-8) and (5-9). Equation (5-7) indicates that there exists a correspondence between $\boldsymbol{s}_t$ and $\boldsymbol{C}(t)$, i. e. , we can obtain the face contour $\boldsymbol{C}(t)$ based on the state space $\boldsymbol{s}_t$.

Using Graphical Model to Model the Relationship among Facial Features

The dynamic model in the previous section is for one facial feature, and we can build several dynamic models, each for one different facial feature. The multiple independent ARP model based tracking method for multiple facial features tends to fail in multiple feature tracking, since the inter-relationships among facial features are not taken into account. Actually, the motions of each facial feature relate to each other. For example, when one frowns, his eyes will become smaller; when one surprises with wide open mouth, the eyebrows will move up. It is difficult to describe this kind of inter-relationship deterministically. In this section, we use probabilistic graphical model—Bayesian network to describe it non-parametrically.

- Bayesian Network Bayesian network is a Directed Acyclic Graph (DAG). The Bayesian network used in this section is shown in Fig. 5. 20,

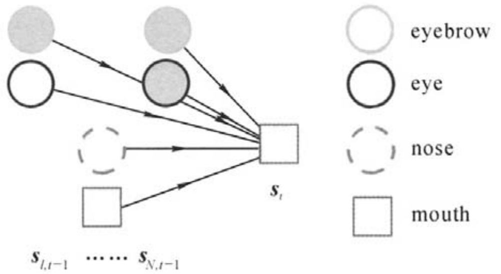


Fig. 5. 20 Bayesian network based dynamic model for multiple facial feature prediction

where the filled circle denotes observation node, and the empty circle denotes hidden node. The directed edge represents the statistical dependency between two nodes, and the direction is from the parent node to the child node. The intuitive meaning of Fig. 5. 20 is that we can predict the current position of mouth's contour on condition that we have already known the positions of each facial feature's contours in the previous time instant.

- Bayesian Network Based Dynamic Model We utilize Bayesian inference to calculate the marginal probability $p(s_{j,t} \mid \{s_{i,t-1}\}_{i=1}^{N})$. For multiple facial feature tracking, the intuitive meaning is to predict the contour state parameter $s_{j,t}$ in current time instant t on condition that each facial feature's contour state parameters $\{s_{i,t-1}\}_{i=1}^{N}$ are already known. The result of prediction is $\hat{s}_{j,t}$ that maximizes the marginal probability:

$$\hat{s}_{j,t} = \arg\max_{s_{j,t}} p(s_{j,t} \mid \{s_{i,t-1}\}_{i=1}^{N}) \tag{5-10}$$

Generally, the Bayesian model based dynamic model cannot be decomposed except that $s_{1,t-1}, \cdots, s_{i,t-1}, \cdots, s_{N,t-1}$ are mutually independent on condition of $s_{j,t}$. But we could use equation (5-11) to approximate the joint marginal probability.

$$p(s_{j,t} \mid \{s_{i,t-1}\}_{i=1}^{N}) = \sum_{i=1}^{N} p(s_{j,t} \mid s_{i,t-1}) \tag{5-11}$$

- Training Bayesian Network Based Dynamic Model Different from the parametric second-order ARP based dynamic model $p(s_{j,t} \mid \{s_{i,t-1}\}_{i=1}^{N})$, Bayesian network based dynamic model is non-parametrical. In order to solve the non-parametric dynamic model, the key point is to calculate $p(s_{j,t} \mid s_{i,t-1})$, From the conditional probability theorem, we have:

$$p(s_{j,t} \mid s_{i,t-1}) = p(s_{j,t}, s_{i,t-1}) / p(s_{i,t-1}) \tag{5-12}$$

where $p(s_{j,t}, s_{i,t-1})$ is joint probability, and $p(s_{i,t-1})$ is the probability of facial feature i in the previous time instant. From the training data, we fit the mixtures of Gaussians to $p(s_{j,t}, s_{i,t-1})$ and $p(s_{j,t})$. We can obtain $p(s_{j,t} \mid s_{i,t-1})$ by evaluating equation (5-12). The sketch maps of $p(s_{j,t}, s_{i,t-1})$ and $p(s_{i,t-1})$ are shown in Figs. 5. 21 and 5. 22 respectively.

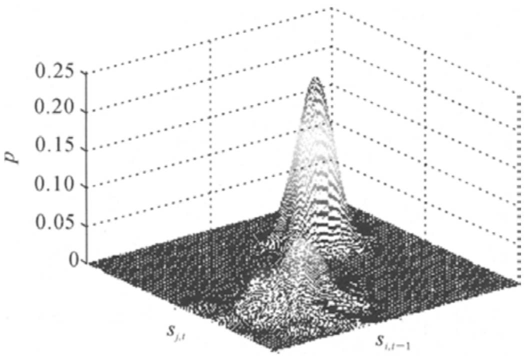


Fig. 5. 21 Sketch map of $p(s_{j,t}, s_{i,t-1})$

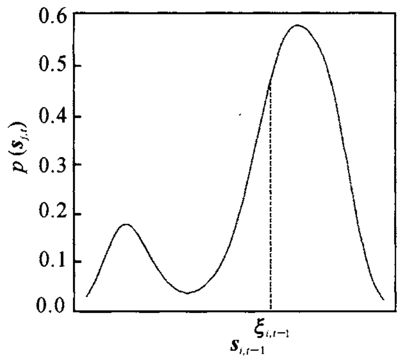


Fig. 5. 22 Sketch map of $p(s_{i,t-1})$

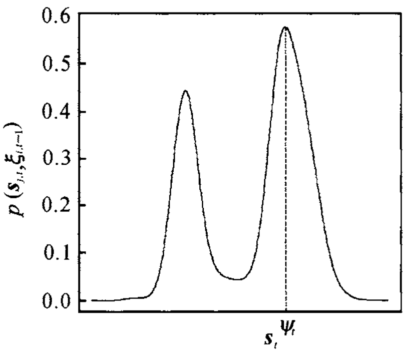


Fig. 5. 23 Sketch map of $p(s_{j,t}, s_{i,t-1})|_{s_{i,t-1}=\xi_{i,t-1}}$

- Using Bayesian Network Based Dynamic Model to Predict Contour of Feature On condition that the facial feature i's state parameter $s_{i,t-1}$ is equal to $\xi_{i,t-1}$ in the previous time instant, we can predict the state of facial feature j based on equations (5-10)—(5-12).

$$\hat{s}_{j,t} = \arg\max_{s_{j,t}} p(s_{j,t} \mid \{\xi_{i,t-1}\}_{i=1}^{N}) = \arg\max_{s_{j,t}} \prod_{i=1}^{N} p(s_{j,t} \mid \xi_{i,t-1})$$
$$= \arg\max_{s_{j,t}} \prod_{i=1}^{N} p(s_{j,t}, \xi_{i,t-1}) / p(\xi_{i,t-1}) \qquad (5\text{-}13)$$

When $s_{i,t-1} = \xi_{i,t-1}$, $p(s_{j,t}, s_{i,t-1})|_{s_{i,t-1}=\xi_{i,t-1}}$ is single variable MOG, its one-dimensional sketch map is shown in Fig. 5. 23. From equation

(5-13), we know that we need to calculate the maximum of the product of N MOGs. Since it is difficult to obtain the maximum directly, practically approximative methods are used, e. g. , starting from an arbitrary point, use gradient descent algorithm to obtain the local maximum; utilizing discretization, draw ns samples, then find the maximum probability of them. We tend to use the latter method, since the global maximum can be obtained.

Low-level Feature Based Face Tracking in Advance

The second order ARP based dynamic model is used for each facial feature, not for the whole face. To avoid the tracked facial feature's contour drifting out of the face, it is necessary to track the whole face firstly. Therefore we could narrow down the search range for the facial feature tracking.

Here we use color histogram based Meanshift algorithm to track the whole face, and obtain the location of human face (see Fig. 5. 24(a),(b)). By this means, we set a search range for the observation model of facial feature tracking. Since observation is the most time-consuming part of the facial feature tracking, narrowing down the search range makes the tracking more efficient. Meanshift tracking algorithm can also be used to obtain

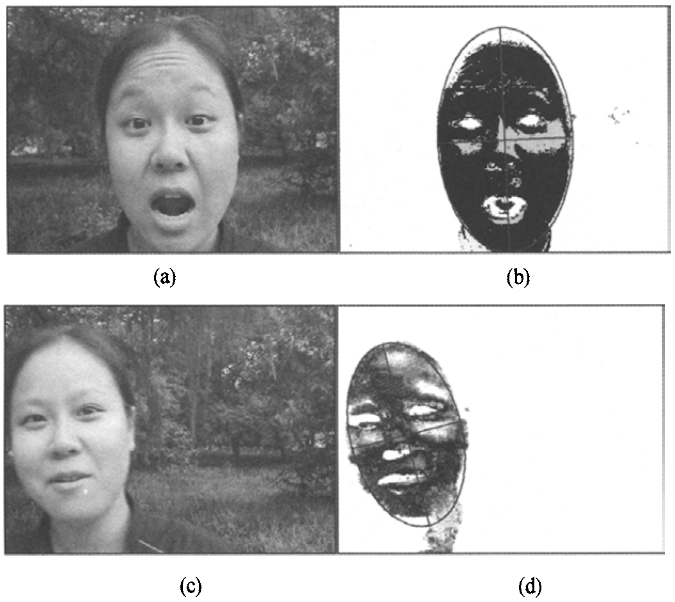


Fig. 5. 24 Meanshift algorithm is used to obtain the face's position and orientation. (a) One frame contains the frontal face; (b) The probabilistic distribution map of face, and the tracked face area is shown in the ellipse; (c) One frame contains the face with orientation; (d) The probabilistic distribution map of face, and the across approximately gives the face's orientation

the orientation of face (see Fig. 5. 24(c),(d)), and this makes preparation for spatial constraint in the following section.

In the experiments, the training for dynamic model was carried out for frontal faces. When the tracked face has orientation, we use Meanshift algorithm to approximately get its orientation, and then warp the face image into the canonical position. We do the facial feature tracking on the canonical face, and undo the warping after tracking for graphical display.

Spatial Constraint Among Facial Features

Human face image belongs to a special class. The facial feature position of different person only varies in a local area. If we know the position and orientation of a face, we can use the spatial constraint among facial features to obtain the approximate position of each facial feature. As shown in Fig. 5. 25(a), the human face can be described by an ellipse. The ratio between length and width is roughly 7 : 5, and the distance between two eyes' centers is about 2/5 of the width of face. For a face with orientation, this spatial constraint still holds (see Fig. 5. 25(b)).

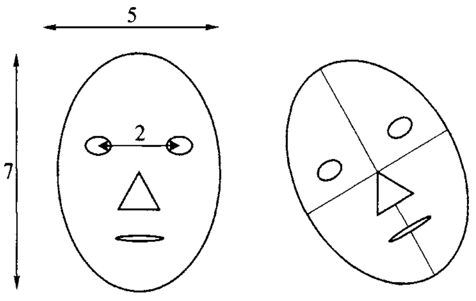


Fig. 5. 25 (a) The spatial constraint among facial features; (b) The spatial constraint also holds for face with orientation

Multi-cue Fusion for Prediction

The spatial constraint among facial features can be combined with face tracking to specify the approximate position of facial features in the current time instant. This kind of low-level prediction can be integrated with the dynamic model based prediction to improve the accuracy of prediction. The low-level Meanshift algorithm based face tracking and spatial constraint among facial features are the preprocessing for prediction, and they can be easily fused into the prediction model.

Integrate Second-order ARP Based Dynamic Model with Graphical Model

Second-order ARP based dynamic model is very quick to predict, but it ignores the influence on the facial feature's position in the current time instant caused by the position of other facial features in the previous time instant. The graphical model based prediction can obtain better result than ARP based method theoretically, but its non-parametric property determines that finding the global maximum is time-consuming. This section combines the advantages of these two methods. The procedure of the algorithm is as follows.

Step 1 Based on equation (5-8), we use reject sampling method to draw ns samples (we choose $ns = 20$ based on experiments) from w_t. By this way, ns ARP based prediction results $s_{j,t}^k$ are generated, where $0 < k < ns$.

Step 2 Substitute $s_{j,t}^k$ for $s_{j,t}$ in equation (5-13), we can find the best prediction $\hat{s}_{j,t}$ from the ns predictions.

Stpe 3 Based on equation (5-7), we can solve the contour $\hat{\boldsymbol{C}}(t)$ of current facial feature in time instant t.

In the ASM/AAM based multiple facial tracking algorithms, their dynamic models are only zero-order or first-order linear models, which can only describe uniform motion or uniform acceleration/deceleration motion. Therefore, the prediction based on these dynamic models is not enough. These tracking algorithms usually converge to correct position only when the initial position of facial feature's contour is reasonably appropriate. If the initial position is not very good, the tracker tends to be locked on a local maximum or fails.

5.2.1.4 Measurement Model

After we have the prediction result of the facial feature's contour position, the result should be verified and adjusted by a measurement model. Compared with the prediction model, the measurement model is relatively easy to construct. On condition that the predicted contour of a facial feature is $\hat{\boldsymbol{C}}(t)$, one measurement in time instant t is to find feature (e. g. edge) along the normal vector $\boldsymbol{n}(pt)$ of one point pt on the contour curve:

$$f(pt,t) = (\boldsymbol{C}(t) - \hat{\boldsymbol{C}}(t)) \cdot \boldsymbol{n}(pt) + \boldsymbol{g}(pt,t) \tag{5-14}$$

where $\boldsymbol{g}(pt,t)$ is an Gaussian noise, and its variance δ^2 is a constant. The visual effect of measurement along the normal vector is to pull the contour curve $\hat{\boldsymbol{C}}(t)$ along the normal direction based on the found feature. Let $\tilde{\boldsymbol{C}}(t)$ denote the contour curve after measurement, and it is the final tracking result in time instant t for current facial feature. After the prediction and measurement model are obtained, they can be integrated into the Kalman filter framework in a standard manner.

5.2.1.5 Experimental Results

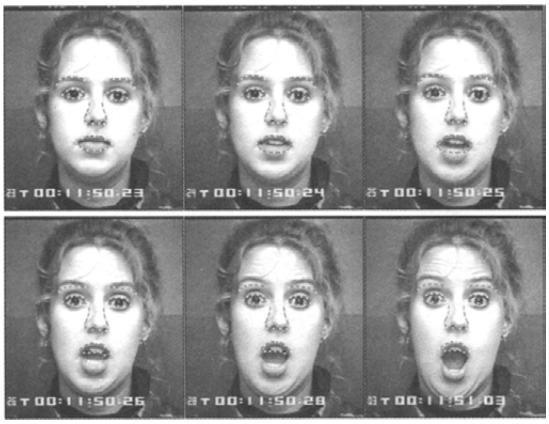
We have implemented a prototype system by Visual C++ and Matlab on Windows platform. We track 6 contours of facial features, which are left eyebrow (right eyebrow), left eye (right eye), nose and mouth. The contours are all quadratic B-spline, where eyebrow and nose are open B-spline, other facial features are described by closed B-spline. The number of control points for eyebrow, eye, nose and mouth is 10, 9, 16, 12 respectively (see Fig. 5. 18). The total number of control points is 66, i. e. , the total dimensionality is 132.

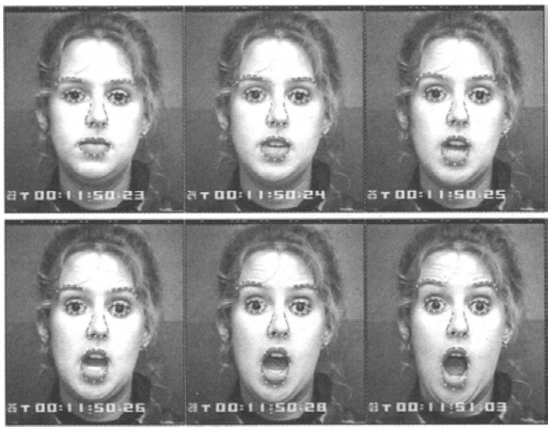
In order to reduce the dimensionality for contour model, we select 100 frames of frontal face images from the training set, and these images belong to 48 different persons. We do PCA for each facial feature. After dimensionality reduction, the dimensionality for eyebrow, eye, nose and mouth is 7, 7, 12 and 9 respectively. The total dimensionality is 49, accounting for 99% variations.

In the training for second-order ARP based dynamic model, we obtain 6 dynamic models from image sequences, each for eyebrows (left and right), eyes (left and right), nose and mouth. In the training, we use interactive editing to manually mark feature points in order to get the ground truth. In the training of Bayesian network based non-parametric dynamic model, we use the same image sequence as the second-order ARP model. For an image sequence with M frames, there are $15(M-1)$ pairs of training data. In other words, there are 15 kinds of data for joint probability $p(\boldsymbol{s}_{j,t}, \boldsymbol{s}_{i,t-1})$, and we fit 8 cluster mixtures of Gaussians to them. For the state $p(\boldsymbol{s}_{i,t-1})$ in the previous time instant, there are 6 kinds of data, we also fit 8 cluster mixtures of Gaussians to them. We can calculate conditional probability $p(\boldsymbol{s}_{j,t} \mid \boldsymbol{s}_{i,t-1})$ from the fitted probabilities. The reason to use mixture of Gaussians is that the relationship between the contour of a facial feature in the previous instant and that in the current time instant is multimodal, and is not Gaussian.

In the experiments, it turns up that facial expressions change very fast, e. g. , it only needs 10 frames to change expressions from neutral to happy (for 30 fps video); therefore there are relatively large motion in adjacent two frames. We carry out two kinds of experiments: (1) tracking multiple facial features in frontal expressive face images in Cohn-Kanade database (640×490, 30 fps) (see Fig. 5. 26); (2) we use digital video camera to capture face image sequence (640×480, 30 fps) with expression, orientation and occlusion in the outside. We track these image sequences (see Fig. 5. 27, zoom in to see clearly). All the tracked image sequences are not included in the training set. We compare our algorithm's result with that of AAM's.

The tracking result for a surprise expression sequence is shown in Fig. 5. 26. We can see from the result of edge detection that, when mouth is wide open, the teeth and tongue form dense cluster for the contour of mouth (see Fig. 5. 26(c)). Active Appearance Model (AAM) is locked on local maximum, since it treats the contour of teeth as that of lower lip, and regards the dark circles as contour of eyebrows. Our algorithm correctly predicts that the mouth will probably open when the eyebrows are rising and the eyes are opening. The original size of image sequence in Fig. 5. 26 is 640×490 pixels, and the tracking is carried out in that size. However for the purpose of display in this section, we crop the image down to the size of 432×490. The frame number is shown in the time code at the bottom of the image.


(a) The tracking result by AAM


(b) The tracking result by proposed approach

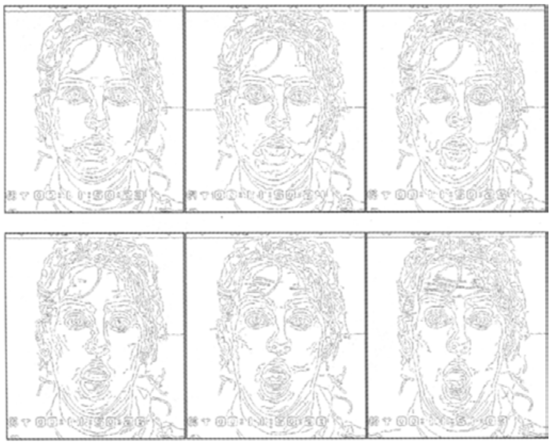
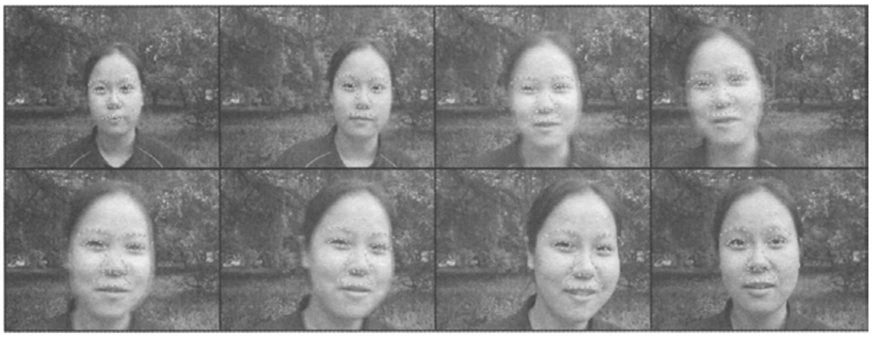
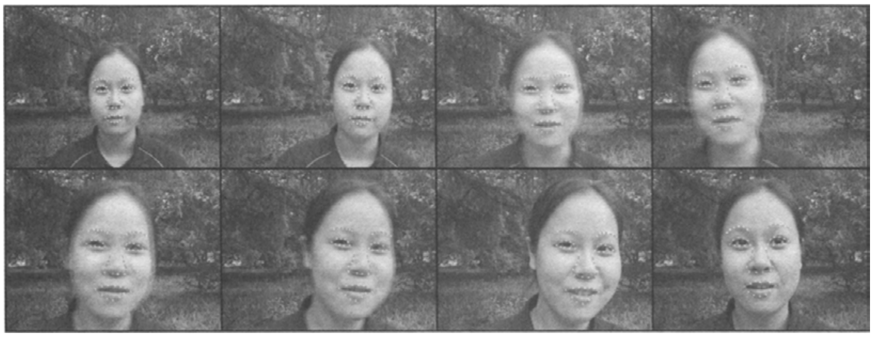

(c) The detection of edges in image sequence

Fig. 5.26 The tracking results of a surprise expression sequence

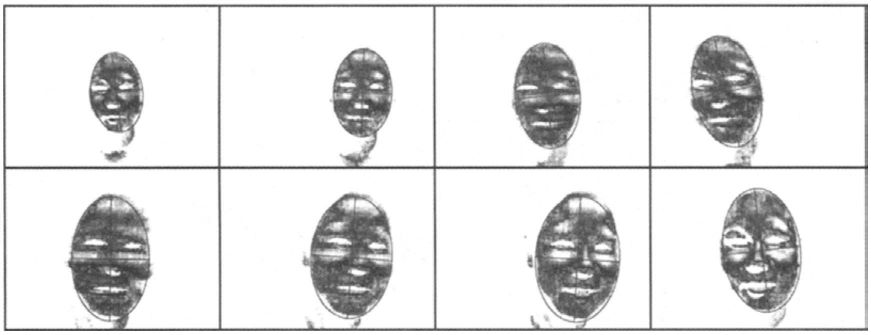
Our algorithm also can robustly track multiple facial features when face has orientation and size variation (zoom in to see Fig. 5. 27(b)). Since we use Meanshift algorithm to get the position of face in advance, we avoid the AAM tracker's problem that left eyebrow is out of the face (zoom in to see Fig. 5. 27(a)). Furthermore, in the graphical model based prediction, we consider the spatial constraint of facial features, and the problem that the contour of upper lip overlaps with that of nose is also avoided. In



(a) The tracking result by AAM



(b) The tracking result by our approach



(c) The pre-tracking result by Meanshift

Fig. 5. 27 Comparison of tracking results: from far to near, quickly approaching the camera, and with head orientation and face expression

Fig. 5. 27, the frame numbers are 9, 37, 42, 62, 70. Our algorithm runs at 3 frames per second on a PentiumIV 1. 8 GHz CPU.

In this section, we introduce a Bayesian network enhanced prediction model based multiple facial feature tracking algorithm. We combine the second-order ARP based dynamic model with the graphical model—Bayesian network based one, and obtain a quick and accurate multi-cue based prediction model. The prediction and measurement model are integrated into the Kalman filter framework in a standard way.

5. 2. 2 Multiple Facial Feature Tracking Based on Probability Graph Model

Probability graph model is an important approach for analysis and inference in computer vision. Multiple facial feature tracking is very challenging because there are plentiful non-rigid motions in facial features besides rigid motions in faces. Non-rigid facial motions are usually very rapid and often form dense cluster by facial features themselves. Only using traditional Kalman filter is inadequate because it is based on Gaussian density, and works relatively poorly in cluster, which causes the density for facial feature's contour to be multi-modal and therefore non-Gaussian. Isard and Blake[11] firstly proposed a face tracker by particle filter CONDENSATION, which is more effective in cluster than Kalman filter.

Although particle filters are often very effective for visual tracking problems, they are specialized to temporal problems whose corresponding graphs are simple Markov chains (see Fig. 5. 28). There is often structure within each time instant that is ignored by particle filters. For example, in multiple facial feature tracking, the expressions of each facial feature (such as eyes, eyebrows, lips) are closely related; therefore a more complex graph should be formulated.

In this section, we employ a spatio temporal graphical model for multiple facial feature tracking (see Fig. 5. 29). Here the graphical model is probability graph model rather than 2D or 3D facial mesh model. In the spatial domain, the model is shown in Fig. 5. 29. Non-parametric belief propagation is used to infer facial feature's inter-relationships in a parts-based face model, consequently the location and state of some features in cluster can be recovered. In temporal domain, every facial feature forms a Markov chain that just likes the conventional CONDENSATION (see Fig. 5. 28).

5.2.2.1 Multiple Facial Feature Tracking by Particle Filter

We adopt the CONDENSATION algorithm to track each facial feature. Although more complex tracker, for example, ICONDENSATION, UPF, and Rao-Blackwellised particle filter can be used, we find the performance of our algorithms to be largely independent of the choice of different particle filter's implementations. So we choose the CONDENSATION for sim-

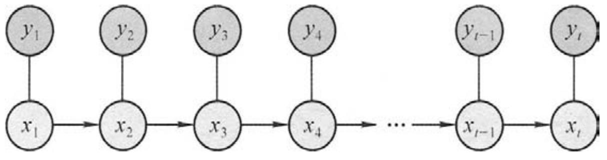


Fig. 5. 28 The Markov chain assumption of particle filters. The empty circle x_t represents the hidden state (contour) in time i, and the filled-in one y_t denotes the local observation

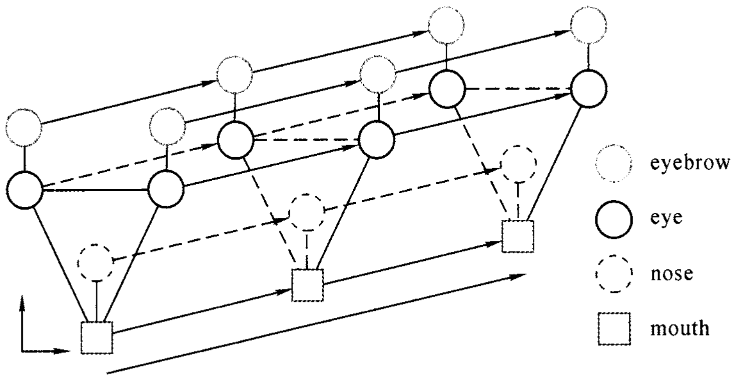


Fig. 5. 29 Tracking multiple facial features with spatio-temporal graphical model. Each facial feature's state (contour) forms a Markov chain in temporal domain, while facial features are related to each other in each time instant

plicity. Six facial features are tracked in this section. They are eyebrows, eyes, nose, and mouth. Take eye for example, we track the eyelid contour. The contour is modeled as B-spline $\boldsymbol{X}_t=\{x_1,x_2,\cdots,x_t\}$, and the observation of eye is $Y_t=\{y_1,y_2,\cdots,y_t\}$. We need to infer the marginal conditional density $p(x_t|Y_t)$. Isard and Blake[12] have proved that:

$$p(x_t|Y_t)=p(x_t|y_t,Y_{t-1})=c_t p(y_t|x_t)p(x_t|Y_{t-1}) \tag{5-15}$$

$$p(x_t \mid Y_{t-1})=\int_{x_{t-1}} p(x_t \mid x_{t-1})p(x_{t-1} \mid Y_{t-1})\mathrm{d}x_{t-1} \tag{5-16}$$

where c_t is constant. In equation (5-15), $p(x_t|Y_{t-1})$ is the effective prior model, and $p(y_t|x_t)$ is the observation model. In equation (5-16), $p(x_t|x_{t-1})$ is the dynamic model.

- Why Several Particle Filters Single particle filter is not suitable to track multiple facial features simultaneously. The reason is as follows: the total dimensionality added by each feature's dimensionality is too high (dozens); The tracking efficiency of particle filter decreases exponentially along with the linear increasing of dimensionality. Usually, it is extremely difficult to get good results from particle filters in spaces of dimensionality much greater than about 10. Even if dimensionality can be reduced by PCA or other nonlinear methods, the total dimensionality of multiple facial features is significantly large. If we reduce the dimensionality too much, valuable state information may be lost.

Human face contains multiple facial features, and it can be decomposed into several parts, such as eyebrows, eyes, nose, and mouth, to form a graphical model in spatial domain. In this section, we track each facial feature by its corresponding particle filter, therefore computational complexity is converted from exponential to linear with the size of the graph.

- Particle Filter Itself is not Enough When there are rapid motions in one facial feature (e. g. mouth) due to the change of facial expressions (see Fig. 5. 30), the corresponding particle filter may fail to track the facial feature's contour. It is difficult to reduce the risk of this failure if the particle filter that tracks only one feature is used. In this section, we track several facial features simultaneously through using several particle filters. When an emotion is presented on the face, different facial features have natural physical interaction. For example, when we smile with blinking left eyes, our left mouth tips will move up; when we surprise with wide-open mouth, the eyebrows will move up.

Instead of constructing heuristic rules for these relationships, we learn the relationships among facial features from training data beforehand. During the process of tracking, if we detect that some facial feature tracker's results are poor, we can infer their positions and states from other facial features by Bayesian inference. In this section, belief propagation is used to carry out Bayesian learning and inference.

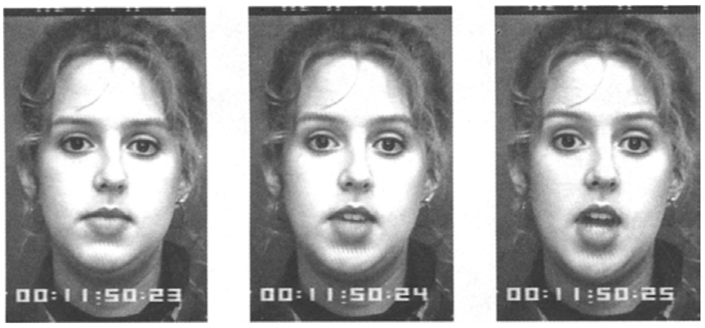


Fig. 5. 30 Three consecutive frames at 30 fps show that facial feature motions are rapid

5.2.2.2 Combining Particle Filter with Belief Propagation

Loopy Belief Propagation

In every time instant, facial features are contained in an undirected graphical model G_f (see Fig. 5. 31). Let V denote the set of nodes (facial features). Nodes are connected by edges E to describe the relationship be-

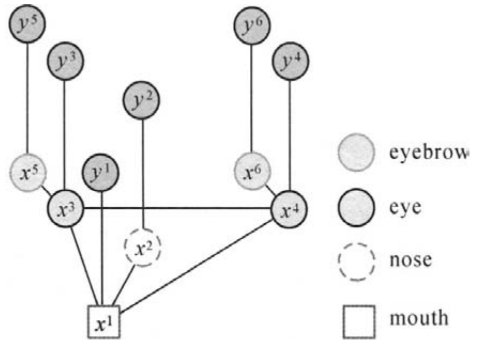


Fig. 5. 31 Markov network representation of a face in spatial domain, x^1, x^2, x^3 (x^4), and x^5 (x^6) denote the contour of mouth, nose, eyes, and eyebrows respectively

tween facial features. The neighborhood of a node i is $NB(i)=\{j\mid(i,j)\in E\}$. Let x^i denote the hidden variable (contour of facial feature), and y^i denote the observed variables (facial feature image). Let $\{x^i\}=\{x^i\mid 1\leqslant i\leqslant N\}$, and $\{y^i\}=\{y^i\mid 1\leqslant i\leqslant N\}$, where N is the number of nodes in graphical model G_t. The joint probability density function factorizes as:

$$p(\{x^i\},\{y^i\})=\frac{1}{C}\prod_{(i,j)\in E}\Psi_{ij}(x^i,x^j)\prod_{i\in V}\phi_i(x^i,y^i) \tag{5-17}$$

where C is a normalization constant, $\Psi_{ij}(x^i,x^j)$ is a correlation function between x^i and its neighbor variable x^j, and $\phi_i(x^i,y^i)$ is an observation function that denotes the evidence of x^i.

From Fig. 5. 31, we can see that it is a Markov network with loops. Experiments motivate us to apply the belief propagation rules in the Markov network with loops. In the belief propagation, we need to calculate the conditional marginal distribution $p(x^i\mid\{y^i\})$ for the nodes that have less confidence of the tracking results by particle filters. The intuitive meaning is that we can infer the facial feature i's position (contour) and state (e. g. expression) from all facial features' observation $\{y^i\}$.

Belief Propagation in Spatio-temporal Graphical Model

In this section, the graphical model is the combination of directed graph (Markov chain) and undirected graph (Markov network). In order to do Bayesian inference, the key point is belief propagation or message passing. The messages of directed graph are passing through the time axis. In Fig. 5. 32, the message passing from x_{t-1}^i to x_t^i is denoted by $M(x_{t-1}^i\rightarrow x_t^i)$. We have:

$$M(x_{t-1}^i\rightarrow x_t^i)=p(x_t^i\mid\{Y_{t-1}^i\}) \tag{5-18}$$

where $\{Y_{t-1}^i\}=\{Y_{t-1}^i\mid 1\leqslant i\leqslant N\}$,

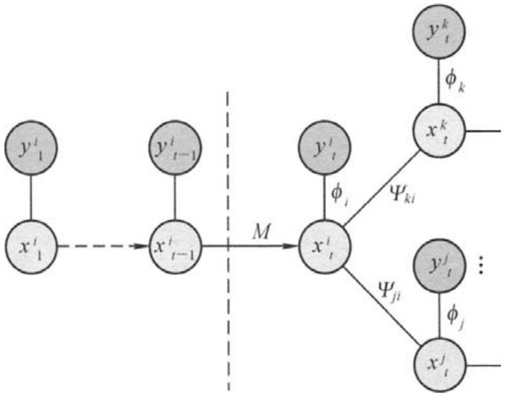


Fig. 5.32 Message passing in a directed-cum-undirected graphical model

$$p(x_t^i \mid \{Y_{t-1}^i\}) = \int_{x_{t-1}^i} p(x_t^i \mid x_{t-1}^i) b(x_{t-1}^i) \mathrm{d}x_{t-1}^i \tag{5-19}$$

and $b(x_t^i)$ is the conditional marginal probability distribution in node (i,t), and it is what we have to calculate. $b(x_{t-1}^i)$ means the tracking result in facial feature i by graphical model in the previous time instant. The belief at node (i,t) is proportional to the product of the local evidence at that node $\phi_i(x_t^i, y_t^i)$, and all the messages coming into it. There are two kinds of messages: one comes from the immediate preceding node x_{t-1}^i temporally; the other is from the neighbors of node (i,t) spatially. Therefore, we have:

$$b(x_t^i) = K\phi_i(x_t^i, y_t^i) M(x_{t-1}^i \rightarrow x_t^i) \prod_{j \in NB(i,t)} m_{ji}(x_t^i) \tag{5-20}$$

where K is a normalization constant and $NB(i,t)$ denotes the nodes neighboring the node (i,t). The message from the previous time is $M(x_{t-1}^i \rightarrow x_t^i)$, as defined in equations (5-18) and (5-19). Furthermore, the message from the spatial neighbor is determined self-consistently by the following message update rule:

$$m_{ji}(x_t^i) = \alpha \int_{x_t^j} \Psi_{ji}(x_t^j, x_t^i) \phi_j(x_t^j, y_t^j) M(x_{t-1}^j \rightarrow x_t^j) \prod_{k \in NB(j,t) \setminus (i,t)} m_{kj}(x_t^j) \mathrm{d}x_t^j \tag{5-21}$$

where we take the product of messages going into node (j,t) except for the one coming from node (i,t). Note that the message $M(x_{t-1}^i \rightarrow x_t^i)$ from the previous time is also considered.

Based on a factorization, we use the observation function $\phi_i(x_t^i, y_t^i) = p(y_t^i | x_t^i)$. We also use the correlation function $\Psi_{ji}(x_t^j, x_t^i) = p(x_t^j, x_t^i) / p(x_t^i)$, and fit this probability with mixtures of Gaussians. The message $M(x_{t-1}^i \rightarrow x_t^i)$ passing from x_{t-1}^i to x_t^i can be viewed as the effective priority: a prediction taken from the marginal probability $b(x_{t-1}^i)$ from the previous time-step, onto which is superimposed one time-step from the dynamical model. From equations (5-20) and (5-21), we can see that ϕ always comes with M. By the analysis above, the product of them is:

$$\phi_i(x_t^i, y_t^i)M(x_{t-1}^i \rightarrow x_t^i) = p(y_t^i \mid x_t^i)p(x_t^i \mid \{Y_{t-1}^i\}) = \frac{1}{c_t}p(x_t^i \mid y_t^i, \{Y_{t-1}^i\}) \tag{5-22}$$

Equation (5-22) means that the product is effectively the posterior probability of x_t^i conditioned on y_t^i and $\{Y_{t-1}^i\}$, and this shares the same idea with the CONDENSATION algorithm. This property is important, because it allows us to firstly run the particle filter to track each facial feature in one time-step, and the output of particle filter is naturally fitted into a loopy belief propagation process (equations (5-20) and (5-21)).

Learning the Correlation Function

In the training database, we manually mark some face's features; therefore we obtain the ground truth position of the contour x^i. First we reduce the dimensionality of facial feature i's contour x^i by PCA. Then from the training data, we fit mixtures of Gaussians to $p(x_t^i)$ and the joint probabilities $p(x_t^j, x_t^i)$ for neighboring facial feature i and j. We evaluate $p(x_t^j \mid x_t^i) = p(x_t^j, x_t^i)/p(x_t^i)$, therefore the correlation function $\Psi_{ji}(x_t^j, x_t^i)$ is obtained.

Optimizing Bayesian Inference for Markov Network

Considering that Bayesian inference using belief propagation costs substantial time, we only initiate it when the particle filter's tracking result is poor, or only infer the facial feature whose result is not so good. For the corresponding particle filter on one facial feature, the tracking result on time t can be described by the moments:

$$E(f(\boldsymbol{x}_t) \mid \boldsymbol{Y}_t) = \sum_{m=1}^{M} \pi_t^{(m)} f(\boldsymbol{s}_t^{(m)}) \tag{5-23}$$

A mean position using $f(\boldsymbol{x}_t) = \boldsymbol{x}_t$ can be utilized for graphical display. Moreover, let $f(\boldsymbol{x}_t) = \boldsymbol{x}_t \boldsymbol{x}_t^{\mathrm{T}}$, we obtain the variance $\sigma_t = E(\boldsymbol{x}_t \boldsymbol{x}_t^{\mathrm{T}} \mid \boldsymbol{Y}_t)$ of the tracking result, and we use the variance as a metric of the tracking quality. For each facial feature, we have σ_t^i, $i = 1, \cdots, N$, where N is the number of all facial features (in this section, it is 6). For the facial features that have larger variances, we determine that their tracking results are worse than others. Therefore belief propagation is carried out to infer the more plausible positions of their contours.

5.2.2.3 Experimental Results

We have developed a prototype system on Windows platform using Visual C++ to implement the algorithms in this section. There are 6 contour models for each facial feature: eyebrows, eyes, nose, and mouth. Each contour is a quadric B-spline curve, in which nose and eyebrows are open curves, and others are closed curves. As shown in Fig. 5.34, there are 6, 9, 12, 12 control points for left (right) eyebrow, left (right) eye, nose, and mouth respectively. The total number of control points is 54; therefore the dimensionality is 108.

We choose Cohn-Kanade facial expression database as the training set, because it contains plenty of frontal faces with rich facial expressions. This database is stored as 30 fps gray-scale image sequences. To learn the relationships among facial features, we have selected 496 frame frontal face images, which belong to 98 different persons, and use interactive program to mark each facial feature's contours. PCA is used to reduce the dimensionality for each facial feature's contour. After that, the dimensionality of left (right) eyebrow, left (right) eye, nose, and mouth is 4, 7, 9, and 9 respectively; therefore the total dimensionality after dimensionality reduction is 40, accounting for 99% of the total variance.

After constructing the PCA bases, we compute the corresponding PCA coefficients for each individual in the training set. Then, for each of facial feature's contour pairs connected by edges in Fig. 5.31, we determine kernel-based non-parametric density estimates for each node itself $p(x_t^i)$ and their joint probabilities $p(x_t^j, x_t^i)$. Fig. 5.33 shows several marginalizations

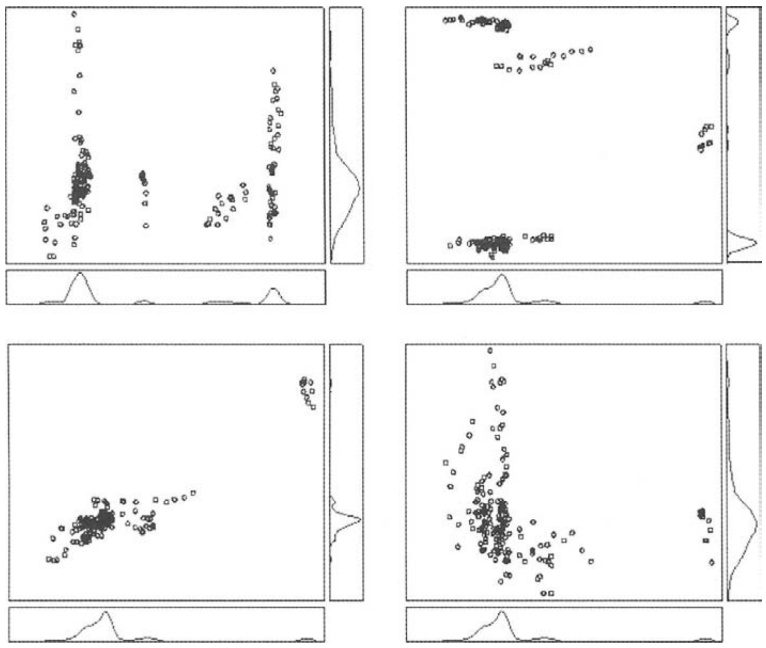


Fig. 5.33 Joint densities of four different pairs of PCA coefficients, it can be seen that the marginal distributions are multimodal

of $p(x_t^j, x_t^i)$, each of which relates a single pair of PCA coefficients (e. g. the first mouth and second left eye contour's coefficients). We can see that simple Gaussian approximations would lose most of this data set's meaningful structure.

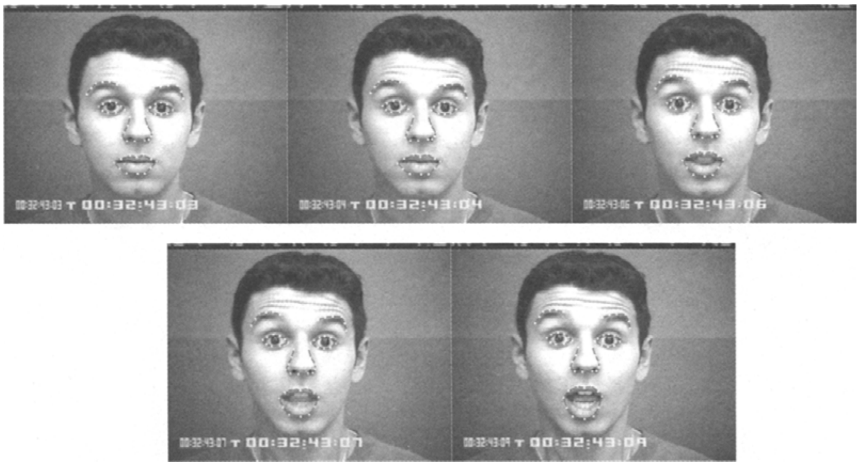
We have also trained the dynamic model for each facial feature. For observation model, a set of independent measurement lines that perpendicular to the hypothesized contour are used to measure the likelihood of detected edge points.

Using a single CONDENSATION tracker with multiple contours to track multiple facial features is infeasible because the dimensionality is much higher than 10. Here we compare our results with those of Multiple Independent CONDENSATION-style Trackers (MICT). We have tested our algorithm on the image sequences in Cohn-Kanade database and the videos (640×480, 30 fps) that we captured by a digital video camera. The test image sequences are not included in the training database. The results (Figs. 5. 34—5. 36) indicate that our tracker is more robust than MICT. Our tracker runs at about 3 fps and the MICT tracker runs at about 4 fps on a PentiumIV 1. 8 GHz CPU.

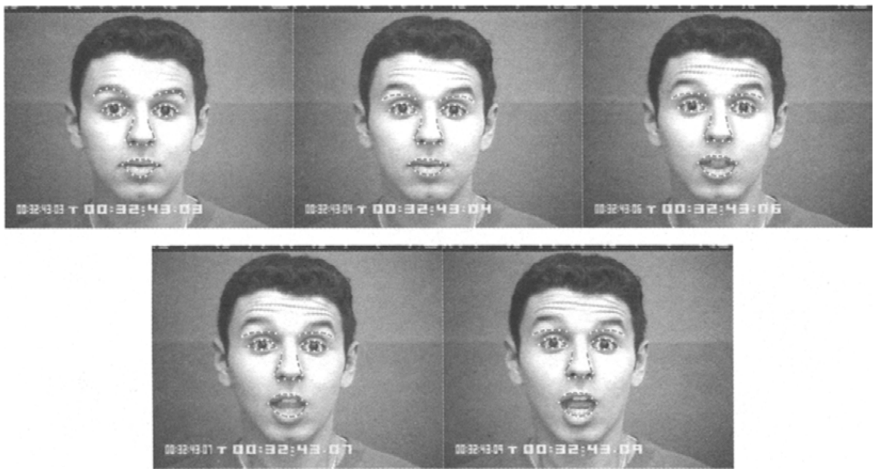
In this section, we extend the particle filter from the relatively simple Markov chain to the directed-cum-undirected graphical model applied to multiple facial feature tracking problem. Spatial structure information and relationships among nodes in each time instant are effectively considered by Bayesian learning and inference in the loopy belief propagation framework.

5. 2. 3 3D Facial Expression Reconstruction

Facial animation is a challenging topic in computer graphics, because facial expression involves non-rigid motion of multiple facial organs. Under current technical condition, only performance driven facial animation technique provides solution to realistic 3D facial animation. In terms of data source, the performance driven animation technique can be divided into two classes: (1) the 3D facial motion data are captured by special motion capture equipment; (2) the 3D facial motion data are obtained via tracking a video sequence. Using motion capture equipments is a direct way to obtain the 3D motion data. The well-known manufacturers are Vicon company and Motion Analysis company. Though the motion capture equipments have been widely used in film making, animation and PC game, the equipment is very expensive to ordinary users. On the contrary, video driven facial animation is attracting people's attention since digital video is available everywhere and it is much cheaper than the motion capture equipment. An indispensable procedure in the video driven facial animation technique is 3D reconstruction of 2D motion data. In Sect. 5. 1, we have obtained the motion trajectory of facial features in video through variant approaches, and the so called 3D reconstruction means converting the 2D tracking data into



(a) Our result

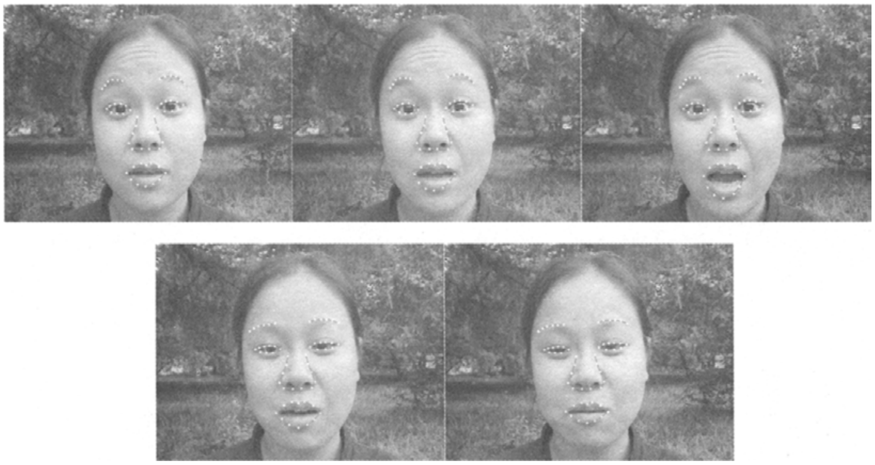


(b) MICT's result

Fig. 5.34 Tracking result of a surprise expression sequence. (a) Our algorithm correctly tracks the eyebrows and mouth; (b) The dark circles and teeth distract the MICT tracker, therefore it fails to track them

3D motion data via computer vision techniques.

Traditional image based 3D reconstruction needs two or more images with different viewpoints. First, the positions of feature points should be aligned on the images, and then the correspondence between the feature points should be established, finally the depth information of feature points can be obtained through stereo vision technique. Generally speaking, calibration is necessary for getting fine results. However, the error that occurred in calibration may cause degeneration of reconstruction results. In many cases, the images are not captured by users themselves,


(a) Our result

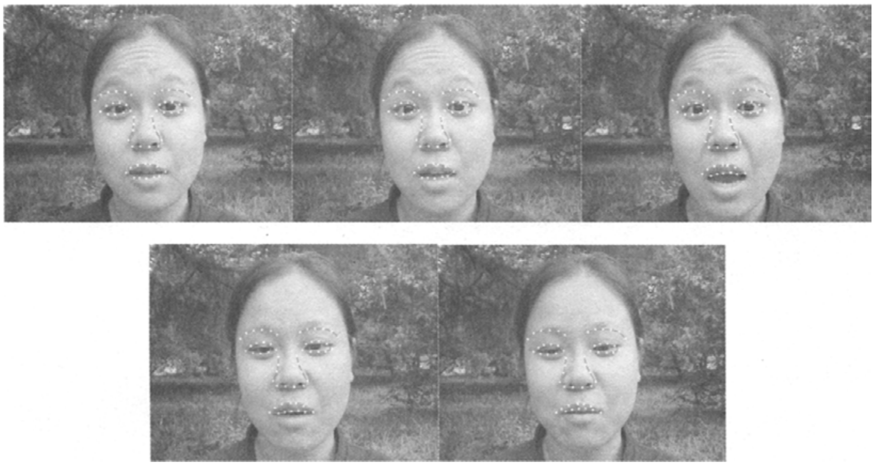
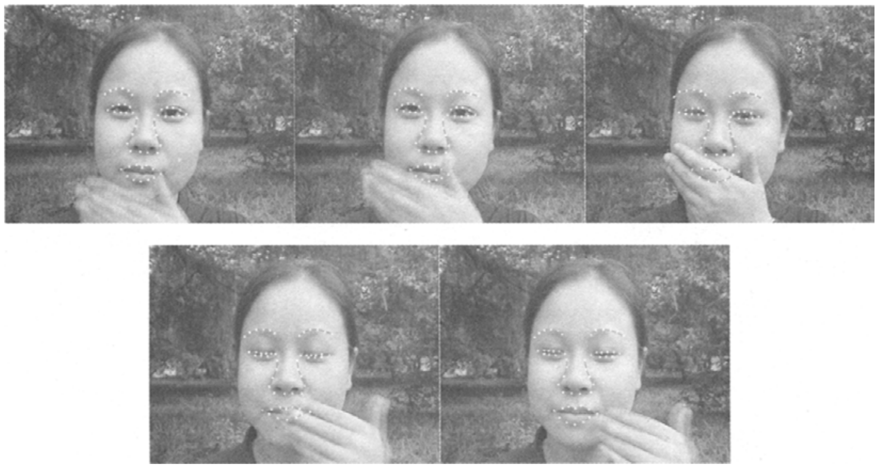
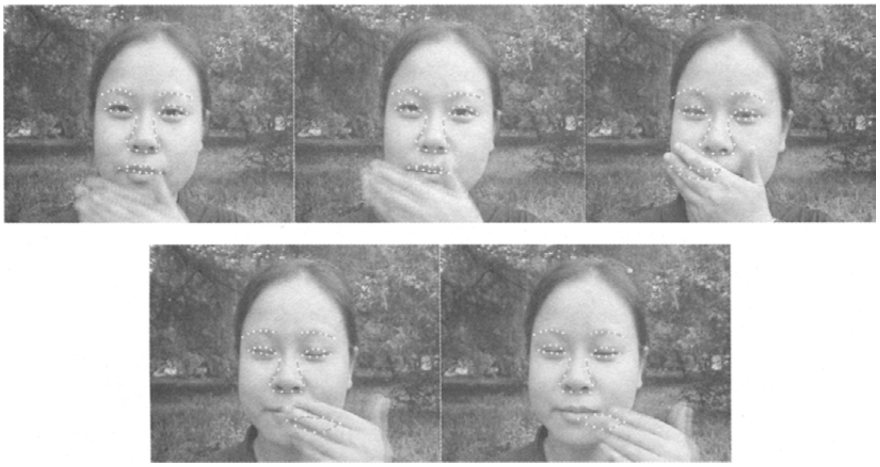

(b) MICT's result

Fig. 5.35 Three consecutive facial expressions: neutral, surprise, and happy. From the first row to second row, and left to right, frame numbers are: 320, 322, 324, 356, 358

thus the inner camera parameters are not available. Therefore, traditional 3D reconstruction is limited by the requirement of pre-calibration. Though researchers have done some 3D reconstruction based on uncalibrated images, these works are restricted to static face modeling, and 3D motion data for driving the face model is still unavailable. So in this section, we will introduce a new approach to reconstruct 3D motion data from uncalibrated monocular video sequence. In this approach, the perspective projection camera model is approximated by weak perspective projection model. We will begin with the basic pinhole camera model.



(a) Our result



(b) MICT's result

Fig. 5. 36 Comparison results of hiding mouth. Frame numbers are: 803, 805, 810, 871 and 872

5.2.3.1 Pinhole Camera Model

Pinhole model is the mostly used camera model owing to its simplicity and accuracy. As shown in Fig. 5. 37, the viewpoint is O, the coordinate of 3D space point $P(X_w, Y_w, Z_w)$ under the camera coordinate system is (X, Y, Z), and the projection on the image plane is $P'(x, y, f)$, we have $f/Z = x/X = y/Y$. Suppose the camera's field of view (fov) along x axis is $xfov$, the resolution is u_x, and the fov along y axis is $yfov$, the resolution is u_y, then the image coordinate (u, v) of P' is:

$$u=\frac{x-x_0}{f\cdot\tan(xfov/2)/u_x}=\frac{x}{d_x}+u_0$$

$$v=\frac{y-y_0}{f\cdot\tan(yfov/2)/u_y}=\frac{y}{d_y}+v_0$$

where d_x, d_y are physical sizes of each pixel along x and y axis, and (u_0, v_0) is the intersection of camera center line and image plane.

Thus we have

$$Z\begin{bmatrix}u\\v\\1\end{bmatrix}=\begin{bmatrix}a_u & 0 & u_0 & 0\\0 & a_v & v_0 & 0\\0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}$$

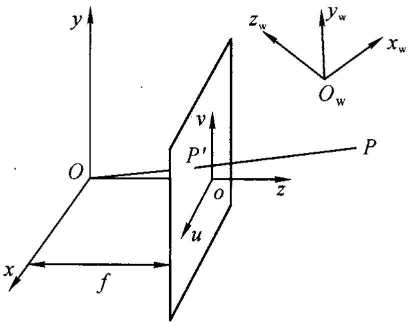


Fig. 5.37 Pinhole camera model

where a_u, a_v are the flex factors along x and y axis. Although a_u, a_v have only 2 degrees of freedom, they are related to multiple variants, therefore the combination of their variance can lead to mathematically equivalent results. For example, d_x, d_y of a real world camera are closely related to f. In this section, fov is used to control the variance. For the sake of simplicity, we regard one camera coordinate system as the world coordinate system, then the transformation to the other camera coordinate system can be described as:

$$\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}=\begin{bmatrix}\boldsymbol{R} & \boldsymbol{t}\\\boldsymbol{0} & \boldsymbol{1}\end{bmatrix}\begin{bmatrix}X_w\\Y_w\\Z_w\\\boldsymbol{1}\end{bmatrix}$$

where (X,Y,Z) and (X_w, Y_w, Z_w) are the camera coordinate and the world coordinate respectively, $\boldsymbol{R}$ is 3×3 rotation matrix and $\boldsymbol{t}$ is translation vector. Thus, the perspective projection of the second image can be described as:

$$Z\begin{bmatrix}u\\v\\1\end{bmatrix}=\begin{bmatrix}a_u & 0 & u_0 & 0\\0 & a_v & v_0 & 0\\0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{R} & \boldsymbol{t}\\\boldsymbol{0} & \boldsymbol{1}\end{bmatrix}\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix}=\boldsymbol{M}\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix}$$

where (u,v) is the projection of the world space point (X_w, Y_w, Z_w) on the second image plane, (u_0, v_0) is the intersection of camera center line and the image plane, here we suppose (u_0, v_0) is the image center. The inner parameters u_0, v_0, a_u, a_v are subject to the field of view along both axis f_x, f_y and the *width* and *height* of the image, i.e., $u_0=width/2$, $v_0=height/2$, $a_u=u_0/\tan(f_x/2)$, $a_v=v_0/\tan(f_y/2)$. The 6 outer parameters belong to $\boldsymbol{R}$ and $\boldsymbol{t}$.

5.2.3.2 Weak Perspective Projection Model

Weak perspective projection is a kind of affine projection which has been widely used in the domain of computer vision. Two steps are needed in the course of weak perspective projection: (1) orthogonally project the space point onto the object plane which is orthogonal to the camera center line; (2) then project the point on object plane to image plane via pinhole perspective model. This procedure is shown in Fig. 5. 38, where $\boldsymbol{P}$ and $\boldsymbol{R}$ are the space points, $\boldsymbol{P}^1$ and $\boldsymbol{R}^1$ are the orthogonal projections on the object plane, then the image points under weak perspective projection are $\boldsymbol{p}^1$ and $\boldsymbol{r}^1$. This procedure can be described by the solid arrow line. The dotted arrow line shows the perspective projection procedure, in which $\boldsymbol{p}$ and $\boldsymbol{r}$ are the perspective projections of $\boldsymbol{P}$ and $\boldsymbol{R}$. The dashed arrow line denotes the orthogonal projection, $\boldsymbol{p}^T$ and $\boldsymbol{r}^T$ are obtained by orthogonal projection of $\boldsymbol{P}$ and $\boldsymbol{R}$. The weak perspective projection is in accordance with orthogonal projection up to a scale factor, and this factor is subject to the distance between the object and the camera optical center O.

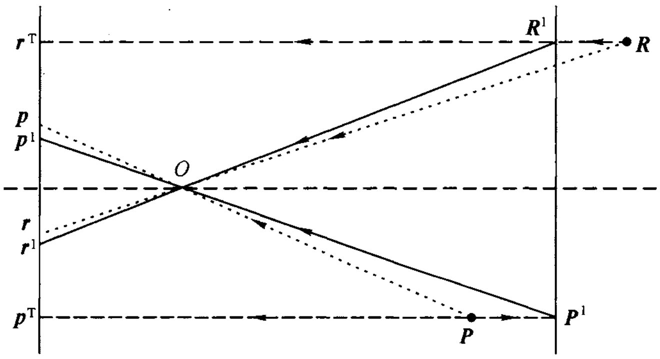


Fig. 5. 38 The weak perspective projection, perspective projection and the orthogonal projection

5.2.3.3 3D Reconstruction by Shape From Motion (SFM) Algorithm under Weak Perspective Projection

According to the rationale of weak perspective projection, when the distance between object and camera is much larger than the depth value of the object itself, weak perspective projection is the reasonable approximation of perspective projection. Human face is in accordance with this condition. According to SFM, non-rigid shape can be seen as a weighted linear combination of a set of shape bases. Given tracking results, 2D features in each frame can be described by weak perspective projection model:

$$\boldsymbol{P}_{fn} = (x, y)^T_{fn} = [e_f c_{f1} \boldsymbol{R}_f, \cdots, e_f c_{fK} \boldsymbol{R}_f] \cdot [S_{1n}, \cdots, S_{Kn}]^T + \boldsymbol{t}_f$$
$$(f = 1, \cdots, F,\ n = 1, \cdots, N) \qquad (5\text{-}24)$$

where F and N are the number of frames and feature points, e_f is non-zero scalar of the weak perspective projection, $c_{f1}, \cdots, c_{fK}$ are combination coefficients of K shape bases $S_{1n}, \cdots, S_{Kn}$, $\boldsymbol{t}_f$ is the translation, $\boldsymbol{R}_f$ stands for the first two rows of the fth camera rotation. We stack the x and y coordinates of each feature in F frames to form a $2F \times N$ tracking matrix $\boldsymbol{P}$, then $\boldsymbol{P}=\boldsymbol{M} \cdot \boldsymbol{S}+\boldsymbol{T}$, where

$$\boldsymbol{M}=\begin{bmatrix} e_1 c_{11}\boldsymbol{R}_1 & \cdots & e_1 c_{1K}\boldsymbol{R}_1 \\ \vdots & \ddots & \vdots \\ e_F c_{F1}\boldsymbol{R}_F & \cdots & e_F c_{FK}\boldsymbol{R}_F \end{bmatrix}, \quad \boldsymbol{S}=\begin{bmatrix} S_{11} & \cdots & S_{1N} \\ \vdots & \ddots & \vdots \\ S_{K1} & \cdots & S_{KN} \end{bmatrix},$$

and $\boldsymbol{T}$ is a translation matrix. We subtract $\boldsymbol{T}$ from $\boldsymbol{P}$ and get the registered tracking matrix: $\bar{\boldsymbol{P}}=\boldsymbol{M} \cdot \boldsymbol{S}$. Then the rank $3K$ approximation of $\bar{\boldsymbol{P}}$ can be determined by performing SVD on $\bar{\boldsymbol{P}}$. After SVD decomposing, we get $\tilde{\boldsymbol{P}}=\widetilde{\boldsymbol{M}} \cdot \tilde{\boldsymbol{S}}$ where $\tilde{\boldsymbol{P}}$, $\widetilde{\boldsymbol{M}}$, $\tilde{\boldsymbol{S}}$ are approximations of $\bar{\boldsymbol{P}}$, $\boldsymbol{M}$, $\boldsymbol{S}$ respectively and K can be determined by rank $(\tilde{\boldsymbol{P}})/3$. In fact, the factorization is not unique, for given any $3K \times 3K$ invertible matrix $\boldsymbol{A}$, $\tilde{\boldsymbol{P}}=\widetilde{\boldsymbol{M}}\boldsymbol{A} \cdot \boldsymbol{A}^{-1}\tilde{\boldsymbol{S}}$ holds. So, if we could uniquely determine $\boldsymbol{A}$, the rotation, shape coefficients and shape bases could be denoted as $\boldsymbol{M}=\widetilde{\boldsymbol{M}} \cdot \boldsymbol{A}$, $\boldsymbol{S}=\boldsymbol{A}^{-1} \cdot \tilde{\boldsymbol{S}}$.

It can be known from Fig. 5. 38 that the weak perspective projection is in accordance with orthogonal projection up to a scale factor, so we use the orthonormality constraints [12] on rotation matrices to recover $\boldsymbol{A}$. Let $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ equal $\boldsymbol{Q}$, then $\boldsymbol{M}\boldsymbol{M}^{\mathrm{T}} = \widetilde{\boldsymbol{M}}\boldsymbol{Q}\widetilde{\boldsymbol{M}}^{\mathrm{T}}$. Represent the ith two rows of $\widetilde{\boldsymbol{M}}$ by $\widetilde{\boldsymbol{M}}_{2 \cdot i-1 : 2 \cdot i}$, we get:

$$\widetilde{\boldsymbol{M}}_{2 \cdot i-1}\boldsymbol{Q}\widetilde{\boldsymbol{M}}^{\mathrm{T}}_{2 \cdot i-1}=\widetilde{\boldsymbol{M}}_{2 \cdot i}\boldsymbol{Q}\widetilde{\boldsymbol{M}}^{\mathrm{T}}_{2 \cdot i} \tag{5-25}$$

$$\widetilde{\boldsymbol{M}}_{2 \cdot i-1}\boldsymbol{Q}\widetilde{\boldsymbol{M}}^{\mathrm{T}}_{2 \cdot i}=\boldsymbol{0} \tag{5-26}$$

However, Xiao, et al. [13] proved that only rotation constraints are inadequate for solving the exact scaled rotation matrix $\boldsymbol{M}$. Since $\boldsymbol{A}$ is an invertible matrix with rank $3K$, they denoted the tth three columns of $\boldsymbol{A}$ as $\boldsymbol{a}_k$, $k=1, \cdots, K$, thus $\boldsymbol{Q}_k=\boldsymbol{a}_k\boldsymbol{a}_k^{\mathrm{T}}$ represents the kth component of $\boldsymbol{Q}$, that is, $\boldsymbol{Q}=\boldsymbol{Q}_1+\cdots+\boldsymbol{Q}_k$. By examining the singular values of each frame's image projection, they selected K frames involving independent shape bases. And for each $\boldsymbol{Q}_k$, based on the independency between shape bases, they developed another set of shape basis constraints.

$$\widetilde{\boldsymbol{M}}_{2 \cdot i-1}\boldsymbol{Q}_k\widetilde{\boldsymbol{M}}^{\mathrm{T}}_{2 \cdot j-1}=\boldsymbol{1} \quad (i,j) \in \omega_1 \tag{5-27}$$

$$\widetilde{\boldsymbol{M}}_{2 \cdot i-1}\boldsymbol{Q}_k\widetilde{\boldsymbol{M}}^{\mathrm{T}}_{2 \cdot j-1}=\boldsymbol{0} \quad (i,j) \in \omega_2 \tag{5-28}$$

$$\widetilde{\boldsymbol{M}}_{2 \cdot i}\boldsymbol{Q}_k\widetilde{\boldsymbol{M}}^{\mathrm{T}}_{2 \cdot j}=\boldsymbol{1} \quad (i,j) \in \omega_1 \tag{5-29}$$

$$\widetilde{\boldsymbol{M}}_{2 \cdot i}\boldsymbol{Q}_k\widetilde{\boldsymbol{M}}^{\mathrm{T}}_{2 \cdot j}=\boldsymbol{0} \quad (i,j) \in \omega_2 \tag{5-30}$$

$$\widetilde{\boldsymbol{M}}_{2 \cdot i-1}\boldsymbol{Q}_k\widetilde{\boldsymbol{M}}^{\mathrm{T}}_{2 \cdot j}=\boldsymbol{0} \quad (i,j) \in \omega_1 \cup \omega_2 \tag{5-31}$$

$$\widetilde{\boldsymbol{M}}_{2 \cdot i}\boldsymbol{Q}_k\widetilde{\boldsymbol{M}}^{\mathrm{T}}_{2 \cdot j-1}=\boldsymbol{0} \quad (i,j) \in \omega_1 \cup \omega_2 \tag{5-32}$$

$$\omega_1=\{(i,j)\,|\,i=j=k\},\ \omega_2=\{(i,j)\,|\,i=1,\cdots,K,\ j=1,\cdots,F,i\neq k\}$$

Combining these two kinds of constraints, we can solve $\boldsymbol{Q}$ correctly, thus, $\boldsymbol{A}$ can be computed through SVD. As discussed above, $\boldsymbol{M}$ can be obtained by $\widetilde{\boldsymbol{M}}\cdot\boldsymbol{A}$, since the scale factor $e_1,\cdots,e_F$ can be seen as constant, the generalized camera projection matrix can be denoted as:

$$\boldsymbol{M}=\begin{bmatrix} c_{11}^1\boldsymbol{R}_1 & \cdots & c_{1K}^1\boldsymbol{R}_1 \\ \vdots & \ddots & \vdots \\ c_{F1}^1\boldsymbol{R}_F & \cdots & c_{FK}^1\boldsymbol{R}_F \end{bmatrix}.$$

Since

$$\boldsymbol{R}_f=\begin{bmatrix} r_{f1} & r_{f2} & r_{f3} \\ r_{f4} & r_{f5} & r_{f6} \end{bmatrix}\quad (f=1,\cdots,F)$$

are the first two rows of camera rotation matrix. We spread the two rows in $\boldsymbol{M}$ corresponding to the fth frame as:

$$\boldsymbol{m}_f=\begin{bmatrix} c_{f1}^1 r_{f1} & c_{f1}^1 r_{f2} & c_{f1}^1 r_{f3} & \cdots & c_{fK}^1 r_{f1} & c_{fK}^1 r_{f2} & c_{fK}^1 r_{f3} \\ c_{f1}^1 r_{f4} & c_{f1}^1 r_{f5} & c_{f1}^1 r_{f6} & \cdots & c_{fK}^1 r_{f4} & c_{fK}^1 r_{f5} & c_{fK}^1 r_{f6} \end{bmatrix}$$

and rearrange the entries to obtain

$$\boldsymbol{m}_f^1=\begin{bmatrix} c_{f1}^1 r_{f1} & c_{f1}^1 r_{f2} & c_{f1}^1 r_{f3} & c_{f1}^1 r_{f4} & c_{f1}^1 r_{f5} & c_{f1}^1 r_{f6} \\ & & & \vdots & & \\ c_{fK}^1 r_{f1} & c_{fK}^1 r_{f2} & c_{fK}^1 r_{f3} & c_{fK}^1 r_{f4} & c_{fK}^1 r_{f5} & c_{fK}^1 r_{f6} \end{bmatrix}$$

This matrix can be seen as the inner product of column vector $[c_{f1}^1,\cdots,c_{fK}^1]^{\mathrm{T}}$ and row vector $[r_{f1},r_{f2},r_{f3},r_{f4},r_{f5},r_{f6}]$. The camera projection matrix and the combination weights for the shape bases of each frame can be solved by applying SVD on $\boldsymbol{m}_f^1$. Accordingly, the 3D shape in Euclidean space can be obtained. The procedure of SFM algorithm adopted in this section can be listed as bellow:

Step 1 Construct 2D motion data matrix $\bar{\boldsymbol{P}}$ with the dimensionality $2F\times K$ when given the tracking results;

Step 2 Apply SVD on $\bar{\boldsymbol{P}}$ and obtain: $\bar{\boldsymbol{P}}=\boldsymbol{U\Sigma V}^{\mathrm{T}}$;

Step 3 Compute the rank $3K$ approximate value of $\bar{\boldsymbol{P}}$ as $\widetilde{\boldsymbol{P}}=\widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}^{\mathrm{T}}$, *where* $\mathrm{K}=rank(\bar{\boldsymbol{P}})/3$;

Step 4 Construct $\widetilde{\boldsymbol{M}}=\widetilde{\boldsymbol{U}}\sqrt{\widetilde{\boldsymbol{\Sigma}}}$ and $\widetilde{\boldsymbol{S}}=\sqrt{\widetilde{\boldsymbol{\Sigma}}}\widetilde{\boldsymbol{V}}^{\mathrm{T}}$;

Step 5 Solve optimal $\boldsymbol{Q}$ according to equations (5-25) and (5-32);

Step 6 Compute the camera projection matrix and the combination weights of each frame based on $\boldsymbol{M}$, then finally solve the 3D coordinate of feature points as: $\boldsymbol{P}_{3\mathrm{D}}=c_{f1}\boldsymbol{S}_1+\cdots+c_{fK}\boldsymbol{S}_K$, $f=1,\cdots,F$.

In this section, we approximate the perspective projection matrix with the weak perspective projection matrix, and reconstruct 3D motion data via SFM algorithm based on the 2D tracking data. Experimental results indicate that the reconstructed motion data is reasonable and can be used to drive 3D face model and synthesize realistic facial expressions. In the following parts, we will first introduce the key techniques to realistic human face modeling, and then the facial expression driven by reconstructed 3D motion data is demonstrated subsequently in Sect. 5. 4. 1.

5.3 Video-based Human Face Modeling Techniques

The performance driven facial animation is a popular way in facial animation synthesis. Realistic 3D facial expression can be obtained by driving the personalized 3D face model with the 3D facial expression motion data. Much work has been done in the domain of 3D realistic human face modeling. The most impressive work was proposed by Blanz and Vetter[14]. Utilizing a 3D face sample base, they derived a morphable face model by transforming the shape and texture of the examples into vector space representation. New faces and expressions can be well modeled by forming linear combinations of the examples. However, the computation of correspondences and face model parameters is burdensome work. When two or multiple viewpoint images are available, stereo vision algorithm can be utilized to achieve the 3D reconstruction. Pighin, et al.[15] extracted facial features from several uncalibrated images and reconstructed a personalized 3D realistic face model. They first selected feature points on these images and computed camera parameters based on the correspondence between feature points on different images, then solved for the 3D coordinates of these feature points. The realistic 3D face model was obtained by deforming a generic 3D face model via scattered interpolation using these personalized 3D feature points. Stereo vision based 3D reconstruction needs images with multiple informative viewpoints, but this is not available in many real-world applications. Therefore, 3D face modeling from monocular video sequences is of great importance.

In this part, we introduce a single frontal image based face modeling approach. We use Locally Linear Embedding (LLE) to search for optimal samples in sample space and learn the reconstruction weights. The personalized 3D face is synthesized by linear combination of these samples, and the texture is synthesized by mapping the frontal image to the 3D face. LLE algorithm has been introduced in detail in Sect. 5. 1. 2, so we focus on the 3D face reconstruction in the following part.

5.3.1 Dimensionality Reduction by LLE

To support the reconstruction work, we use 3D head samples generated by 3D sculpture software FaceGen Modeller. The database includes 55 males and 45 females, ranging in age from 25 to 50, in which 80 are eastern Asian and 20 are Caucasian. After preprocessing for good performance, each head model has 6,174 vertices which constitute 6,054 quadrangles. What is more, the topology relationships of these vertices as well as the point by point correspondences between different head samples are already known during the preprocessing.

According to our work, personalized 3D face reconstruction needs not only the features selected from 2D image but also the statistical information from the 3D face samples. Each 3D face sample could be seen as a high-dimensional datapoint in nonlinear embedded subspace. Manifold learning technique contributes to learning corresponding low-dimensional coordinate in a manifold space. These low-dimensional data reflect the most intrinsic properties of the original samples. LLE algorithm indicates that high-dimensional data and low-dimensional coordinates can be reconstructed by their neighbors with the same weight [9]. So, once the low-dimensional coordinate of the image subject is synthesized by that of the samples, corresponding high-dimensional data can be reconstructed by linear combination of these samples.

5.3.2 Active Shape Model (ASM) and Active Appearance Model (AAM)

Extracting facial features from single image is an indispensable procedure in image-based face modeling. Active Shape Model (ASM) and Active Appearance Model (AAM) are prevalent approaches to the problem of sparse feature alignment. Point Distribution Model (PDM) is used in both approaches.

ASM combines local texture matching and global shape subspace constraint together, and converges to an optimal result by alternant iteration between local search and global constraint. To better learn the shape variance of the training data set, we need to normalize the training images by image rotation, translation and scaling so as to approach a benchmark shape. The shapes are represented by a set of shape vectors. Then PCA is applied on the normalized shape vectors, the workflow can be described as follows.

Step 1 Compute the covariance matrix of the normalized shape:

$$\boldsymbol{\Sigma}_s = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{S}_i - \bar{\boldsymbol{S}})(\boldsymbol{S}_i - \bar{\boldsymbol{S}})^{\mathrm{T}}$$

Step 2 Compute the eigenvalues $\boldsymbol{\Lambda}_s(\lambda_1, \lambda_2, \cdots, \lambda_m)$ according to the following equation:

$$\boldsymbol{\Sigma}_s\boldsymbol{\phi}_s = \boldsymbol{\phi}_s\boldsymbol{\Lambda}_s$$

Step 3 Select corresponding eigenvectors as: $\boldsymbol{P}=(\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_t)$. Since the eigenvectors $\boldsymbol{p}_i$ corresponding to the larger eigenvalues λ_i denotes the most important shape variance, any shape vector can be represented by the k leading eigenvectors as: $\boldsymbol{x}=\bar{\boldsymbol{x}}+\boldsymbol{Pb}$ where $\boldsymbol{b}=(b_1, b_2, \cdots, b_t)^{\mathrm{T}}$ are coefficients for controlling the shape variance. Different shape has different $\boldsymbol{b}$, as is shown in Fig. 5. 39. Since $\boldsymbol{b}$

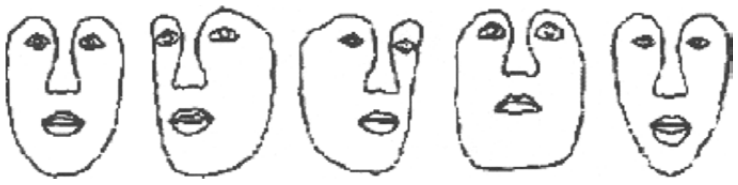


Fig. 5. 39 Different $\boldsymbol{b}$ denotes different shape

is positive definite, $\boldsymbol{b}=\boldsymbol{P}^{\mathrm{T}}(\boldsymbol{x}-\bar{\boldsymbol{x}})$.

In fact, AAM is a further development of ASM, it considers both shape and texture information, and the combination of shape and texture is so called "appearance". It has been widely used in many applications, e. g. facial template alignment, face recognition and face image synthesis. The rationale is to model the variance of shape and texture information via subspace analysis. Suppose $\boldsymbol{W}=\{(\boldsymbol{S}_0, \boldsymbol{T}_0)\}$ where $\boldsymbol{S}_0=[(x_1, y_1), \cdots, (x_k, y_k)]$ is the shape vectors, each shape vector includes K discrete feature points, and $\boldsymbol{T}_0$ is the texture enclosed by $\boldsymbol{S}_0$. We set the average shape $\boldsymbol{S}$ as the benchmark shape and align all the shapes to $\boldsymbol{S}$, thus the average texture of $\boldsymbol{T}_0$ can also be obtained. In AAM, the shape can be represented by K pattern through PCA, in detail, a shape can be represented by a vector $\boldsymbol{b}_{\mathrm{s}}$ ($\boldsymbol{b}_{\mathrm{s}}$ is in fact the model parameter) in feature subspace F. Namely:

Shape is described as: $\boldsymbol{S}=\bar{\boldsymbol{S}}+\boldsymbol{\phi}_{\mathrm{s}}\boldsymbol{b}_{\mathrm{s}}$

Texture is described as: $\boldsymbol{T}=\bar{\boldsymbol{T}}+\boldsymbol{\phi}_{\mathrm{t}}\boldsymbol{b}_{\mathrm{t}}$

The "appearance" of each sample can be described as:

$$\boldsymbol{b}=\begin{bmatrix} \boldsymbol{w}_{\mathrm{s}}\boldsymbol{b}_{\mathrm{s}} \\ \boldsymbol{b}_{\mathrm{t}} \end{bmatrix}=\begin{bmatrix} \boldsymbol{w}_{\mathrm{s}}\boldsymbol{\phi}_{\mathrm{s}}^{\mathrm{T}}(\boldsymbol{S}-\bar{\boldsymbol{S}}) \\ \boldsymbol{\phi}_{\mathrm{t}}^{\mathrm{T}}(\boldsymbol{T}-\bar{\boldsymbol{T}}) \end{bmatrix}$$

where $\boldsymbol{w}_{\mathrm{s}}$ are the weights. Since the metric of shape is distance value and the metric of texture is gray-scale value, they cannot be combined directly. A weight is thus introduced to combine shape and texture information together. The weight is computed as the ratio of the summed eigenvalue of the shape and texture models. Further perform PCA on two feature subspaces and compute the composite appearance model as:

$$\boldsymbol{b}=\bar{\boldsymbol{b}}+\boldsymbol{\phi}\boldsymbol{c}$$

where $\boldsymbol{c}$ is the eigenvectors of the appearance subspace. The rationale of AAM can therefore be described as an optimization problem as bellow: use the model synthesized image to approximate the object image, and adjust the model according to the error between two images.

5. 3. 3 3D Face Modeling

In 5. 3. 1, we have preprocessed the 3D sample faces by dimensionality reduction process. This process is performed offline just once during training, then 3D face modeling can be done based on the dimensionality reduced data. Suppose that the manifold coordinate of sample faces after dimensionality reduction is $\boldsymbol{Y}_{\text{origin}}$, the face modeling algorithm can be described as the following three steps.

Step 1 Automatically align m facial features $\boldsymbol{Fp}$ from the input image by AAM, the facial features are shown in Fig. 5.40(b). After translation, rotation and scaling, $\boldsymbol{Fp}$ is transformed into sample space. Transformed image feature points are denoted as $\boldsymbol{Fp}'$. From $\boldsymbol{Fp}'$, we extract X, Y coordinates of m feature vertices from all face samples. This is done by measuring the Euclidean distances between $\boldsymbol{Fp}'$ and vertices of each face sample. The extracted features are represented as subspace of the original face samples:

$$\boldsymbol{S}_{\text{f}}=\begin{bmatrix} X_{1,1} & \cdots & X_{N,1} \\ Y_{1,1} & \cdots & Y_{N,1} \\ \vdots & \ddots & \vdots \\ X_{1,m} & \cdots & X_{N,m} \\ Y_{1,m} & \cdots & Y_{N,m} \end{bmatrix}$$

$\boldsymbol{S}_{\text{f}} \in \mathbf{R}^{2m\times N}$. Then, we use LLE algorithm to select k_i samples from $\boldsymbol{S}_{\text{f}}$ which best reconstructs $\boldsymbol{Fp}'$. These samples are denoted as $\boldsymbol{S}_i \in \mathbf{R}^{2m\times k_i}$.

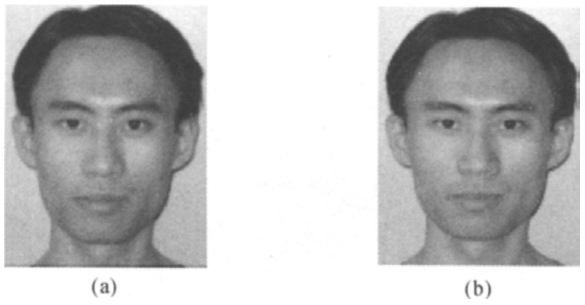


Fig. 5.40 (a) Input frontal face image of one subject; (b) Automatically aligned face image for reconstruction

Step 2 Use LLE algorithm to compute weight $\boldsymbol{W}_{\text{extract}}$ for reconstruction of $\boldsymbol{Fp}'$ by $\boldsymbol{S}_i$. That is, $\boldsymbol{W}_{\text{extract}}$ satisfies $\boldsymbol{Fp}'=\boldsymbol{S}_i\times\boldsymbol{W}_{\text{extract}}$. The computation of weight can be described as:

$$\boldsymbol{W}_{\text{extract}}=LLE([\boldsymbol{Fp}',\boldsymbol{S}_i],k_i,d)$$

where k_i is the neighborhood size computed in Step 1, d is the size of low-dimensional coordinate. Then, given $\boldsymbol{S}_i$, we find the corresponding low-dimensional coordinates $\boldsymbol{Y}'_{\text{origin}}$ from $\boldsymbol{Y}_{\text{origin}}$ through one by one correspondence. Thus, the low-dimensional coordinates of the input image subject can be synthesized by linear weighted combination of $\boldsymbol{Y}'_{\text{origin}}$.

$$\boldsymbol{Y}_{\text{recon}} = \boldsymbol{Y}'_{\text{origin}} \cdot \boldsymbol{W}_{\text{extract}}.$$

In Step 3, $\boldsymbol{Y}_{\text{recon}}$ will be used to reconstruct personalized 3D face geometry from image.

Step 3 Select K_r coordinates from $\boldsymbol{Y}_{\text{origin}}$ which best reconstruct $\boldsymbol{Y}_{\text{recon}}$. These coordinates are represented as $\boldsymbol{Y}_r$. Similarly, given $\boldsymbol{Y}_r$, we can find the corresponding original face samples $\boldsymbol{S}_r$ because of the one by one correspondence between high- and low-dimensional data.

Now, we use LLE algorithm with neighborhood size K_r to get the reconstruction weight $\boldsymbol{W}_{\text{recon}}$ which satisfies $\boldsymbol{Y}_{\text{recon}} = \boldsymbol{Y}_r \cdot \boldsymbol{W}_{\text{recon}}$.

The computation of weight is described as:

$$\boldsymbol{W}_{\text{recon}} = LLE([\boldsymbol{Y}_{\text{recon}}, \boldsymbol{Y}_{\text{origin}}], K_r, d)$$

According to [9], $\boldsymbol{W}_{\text{recon}}$ is also suitable for reconstructing high-dimensional data points in nonlinear space. Thus, the personalized 3D face geometry $\boldsymbol{S}_n$ is reconstructed from a single image as:

$$\boldsymbol{S}_n = \boldsymbol{S}_r \cdot \boldsymbol{W}_{\text{recon}}$$

The reconstructed personalized 3D face model is shown in Fig. 5.41.

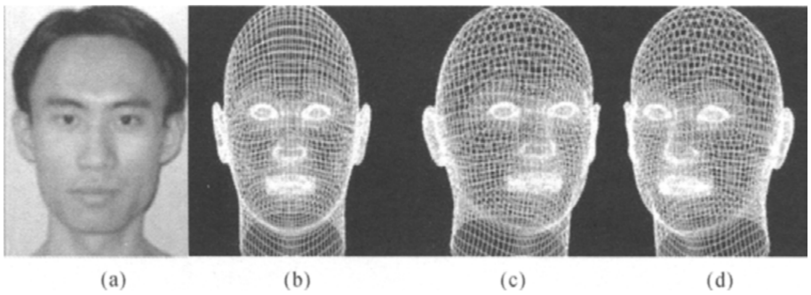


Fig. 5.41 (a) The input frontal face image; (b), (c), (d) are reconstructed 3D face from different viewpoints

5.3.4 Constraint-based Texture Mapping

By nature, texture mapping is a parameterized process of curved surface, namely, finding the corresponding 2D texture coordinates of 3D vertices:

$$S(x_i, y_i, z_i) = [u_i(x_i, y_i, z_i), v_i(x_i, y_i, z_i)] \quad (i=1, \cdots, n)$$

where S is the mapping function, (x_i, y_i, z_i) is the 3D coordinate of a vertex, (u_i, v_i) is texture coordinate of a vertex, n is the number of vertices.

In order to build the photorealistic human face, we first manually assign texture coordinates to some feature vertices of newly reconstructed face geometry. The correspondence between feature coordinates and feature vertices is defined offline as constraint. The texture coordinates can be obtained by our previously built AAM in 5.3.2 and are described as $\boldsymbol{T}_i\{u_i, v_i\}_{i=n}^{l} \subset \mathbf{R}^2$, the corresponding feature vertices on the reconstructed face geometry are depicted as $P_i\{x_i, y_i, z_i\}_{i=1}^{n} \subset \mathbf{R}^3$. To obtain a mapping function from vertices to image coordinates, we use $\boldsymbol{T}$ and $\boldsymbol{P}$ to train RBF network:

$$\boldsymbol{S}(\boldsymbol{P}) = \sum_{i=1}^{n} \alpha_i \phi(P - \boldsymbol{P}_i) = \boldsymbol{T}$$

Once $\boldsymbol{\alpha}$ is known, we obtain the texture coordinates of other vertices by RBF interpolation:

$$\boldsymbol{T}_{\text{new}} = \sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{S}_{\text{new}} - \boldsymbol{S}_{\text{new},\,i})$$

where $\boldsymbol{S}_{\text{new}} = [X_1, Y_1, Z_1, X_2, Y_2, Z_2, \cdots, X_n, Y_n, Z_n]^{\mathrm{T}} \in \mathbf{R}^{3n}$ is the reconstructed 3D face geometry using the method discussed in Sect. 5.3.3.

Texture information between face and ears is unavailable in the frontal image. We solve this problem by sampling from face margin region on the frontal image for RBF interpolation. The texture mapped face as shown in Fig. 5.42 looks fine and natural.

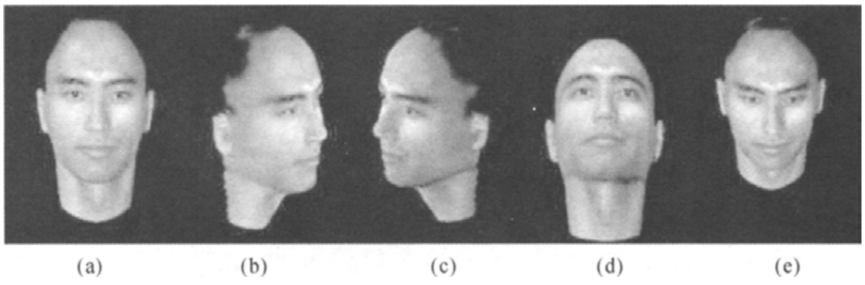


Fig. 5.42 Textured 3D realistic face under different viewpoints

5.3.5 Results and Discussions

We implement our reconstruction algorithm on a PentiumIV 2.4 GHz CPU with 512 M RAM. To reconstruct the face geometry, 52 feature points on the input image are aligned by a well trained AAM and in terms of texture mapping, the feature points also act as constraints. We choose multi-quadric function as basis function and implement the reconstruction process on arbitrary images. The reconstruction results are depicted in Fig. 5.43.

The first row shows the input images, the second to the fifth rows show the 3D models under different viewpoints. The first two images in three test images are captured in our lab, and the third image is obtained from the Internet. Though the face pose in the third image is not so preferable, our approach still gets fine result.

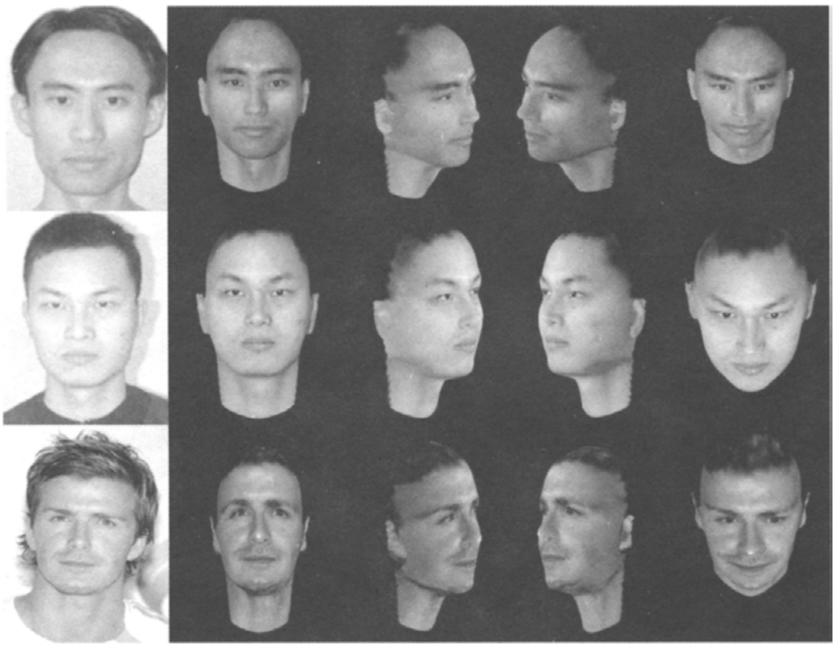


Fig. 5. 43 The reconstructed personalized realistic face models

In this section, we propose a 3D face reconstruction approach. First, LLE is used to reconstruct the 3D geometry, then constraint based texture mapping is applied to do the texture mapping. In brief, the approach has the following advantages: (1) Only a single frontal face image is required for face reconstruction and this is easily satisfied; (2) The system is highly automatic and the accuracy of reconstruction is higher than that of other 3D reconstruction approaches; (3) Constraint based RBF texture mapping provides natural appearance for reconstructed face.

5. 4 Facial Expression Driven Technique

Realistic facial animation is highly important in the computer graphics field as it is an essential facility for human-computer interface and virtual reality and is also a difficult task because there are so many non-rigid motions besides rigid motion of heads as expression changes. Instead of modeling all the complicated facial motions, data-driven facial animation just exploits facial motion data captured in real scenes. Most motion capture systems rely upon the placement of markers on the surface of the face, and the use of vision algorithms to track the movement of those points over time. The tracked feature points are then used to "interpolate" the whole facial mesh. Traditional scattered data interpolation techniques such as RBF or

B-splines are effective for mesh deformation and reconstruction problems with a small part of data loss [16,17]. However, when deforming a face mesh using these methods, significant artifacts may occur because these methods take advantage of only the 3D positions of the mesh vertices but throwing away the information on edges. For example, when the markers on the outer contour of lips move outwards to indicate opening mouth, the lips of the deformable model may be stretched as if they were becoming thicker instead of mouth being opened. On the other hand, the noises in the data, which are inevitable in motion capture and 3D scans, are also not properly considered in such techniques.

In this section, we introduce a solution to data-driven facial animation named manifold Bayesian regression. Our goal is to couple temporally dense facial motion data and a spatially dense static model to provide high resolution in both temporal and spatial domain. Facial motion capture data indicate the movement of marker points on the face mesh. Bayesian regression is like an interpolation method which is trying to find the movement of other points and additionally to smooth out noise. The "interpolated" movement is affected by both the movement of markers and the distance between the point itself and the markers. Euclidean distance is mostly used to measure the nearness between two certain vertices. However, it provides little knowledge about the connectivity of mesh vertices, which is essential in the deformation of meshes. In this section, the geodesic manifold distance [5] is used to replace the Euclidean distance as a novel distance metric in facial deformation algorithms. A static face model can be regarded as a manifold, which is a topological space that is locally Euclidean. The geodesic distance, or the shortest path, is a widely accepted concept in manifold learning to provide spatial understanding of nonlinear topology. We put facial animation into the framework of Bayesian regression. Bayesian approaches provide an elegant way of dealing with noise and uncertainty.

5.4.1 Data Driven Facial Animation

5.4.1.1 Motion Data Preprocessing

Three-dimensional facial motion data may be generated by motion capture devices or reconstructed from two-dimensional motion data tracked in video streams. The face models can be obtained via 3D scanner or exported from modeling software. Highly realistic faces can be modeled from video or images [18]. The face models and facial motion data may be from notably different scenarios and must be aligned in the first step.

Approximate affine transformations are computed and applied to facial motion data at each frame with respect to the correspondences between markers and face mesh vertices so as to align the three-dimensional proportion and alleviate the structural disparity between the motion data and the face model, as shown in Fig. 5. 44. In motion capture context, several markers will usually be added on the top of head for the purpose of global motion estimation, so that separation of the rigid movement and non-rigid expressions can be faster.

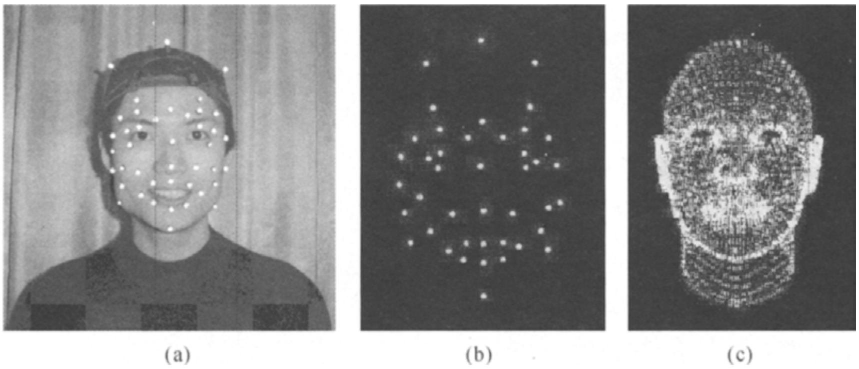


Fig. 5. 44 Facial motion data capture and driving. (a) The actor with 42 markers decorated in the face; (b) Motion capture data captured by Hawk System; (c) Deformable face model represented by 3D points, the points are the mapped motion data

The concept of warping kernels was first proposed by Williams [19] as deformation initiators or feature points to drive facial animations. Here we refer to the facial motion markers which have been already mapped onto the neutral mesh as warping kernels to provide pivots of regressions. The problem of facial deformation can be formulated as estimating data approximate functions $F(\boldsymbol{s},\boldsymbol{t},\theta)$ in nonlinear regression with sparse warping kernels, where $\boldsymbol{s}$ is an input vector indicating the original 3D coordinates of a vertex, $\boldsymbol{t}$ is a vector of the deformed results, and θ is the hyper-parameter which must be learned from the sparse warping kernels. The approximate functions are estimated at each frame of facial motion data to generate sequential facial animation.

5.4.1.2 Geodesic Manifold Distances

Traditional regression techniques such as neural networks and radial basis functions favor the Euclidean distances for measuring the similarity between vertices in input space. However, the Euclidean distances are not suitable for modeling data distributed on complex geometry and topology such as human faces. As shown in Fig. 5. 45, points on different sides of lips are close in the Euclidean metric (straight line segment on the right figure), but far away from each other in its actual topology and kinematic properties.

Nonlinear manifold modeling techniques were developed recently during

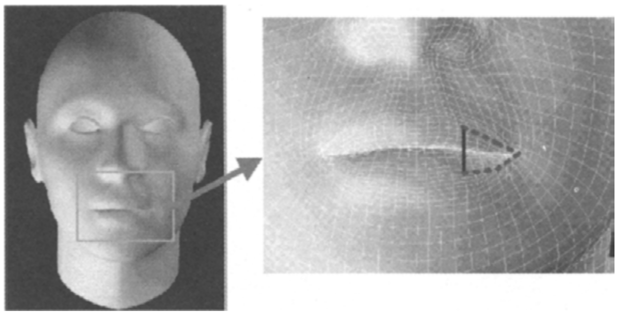


Fig. 5. 45 The Euclidean distance (straight line segment on the right figure) is not suitable for modeling distance between points on different sides of discontinuities. The geodesic distance (dot folding line segments on the right figure) is proved to be a better approach to measure distances in globally non-Euclidean spaces

the research on subspace learning and face recognition, including ISOMAP, LLE and Laplacian Eigenmap (LEM). ISOMAP [5] performs nonlinear dimensionality reduction by applying Multi-dimensional Scaling (MDS) on the geodesic distance matrix. LLE [9] and LEM [20] are local algorithms that represent nonlinear manifold by focusing on the preservation of local neighbour structure. Geodesic manifold distance, which is computed as the shortest path, is a widely accepted concept in the field of manifold learning. A face mesh can be regarded as a manifold, which is a topological space that is locally Euclidean. Points may appear deceptively close by measuring their straight-line Euclidean distance, however far apart on the underlying manifold, as measured by their geodesic manifold distances.

We exploit the geodesic manifold distances to explore the complex geometry and topology of human face models. We take the given face mesh as an undirected graph, where its nodes and arcs are represented by mesh vertices and edges respectively. The sparse matrix of the constructed graph has 22,984 non-zero entries in the 5,832×5,832 matrix in the neutral quadrangle mesh, i. e. only 0. 068% non-zero entries. The geodesic distances are then computed as the shortest path δ_{ij} between nodes $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ in the graph:

$$\delta_{ij} = dijk(\boldsymbol{v}_i, \boldsymbol{v}_j)$$

Furthermore, we construct a distance map from feature points to all other points on the neutral mesh and stored for later expression generations for acceleration. The geodesic distance can be intuitively adopted to extend traditional regression processes such as RBF by replacing Euclidean distance γ with geodesic distance δ in the approximate function, e. g. RBF approximate function with multi-quadrics:

$$F(\boldsymbol{x}_j, \boldsymbol{W}) = \sum_{i=1}^{N} w_i \sqrt{(\gamma_{ji}^2 + \sigma_i^2)}$$

where $\boldsymbol{x}_j$ is a vector representing the 3D coordinates of the input point, $\boldsymbol{W}=[w_1, w_2, \cdots, w_n]^{\mathrm{T}}$ are parameters, γ_{ji} is the Euclidean distance between the feature point and the input point, and σ_i is the stiffness constant that regulates the local or global effects of the feature points. The geodesic distances can accelerate the generation of facial animation by automatic modeling of the discontinuities, which is one of the key problems of data driven facial animation. Traditional facial animation methods facilitate affection volume masks, which use virtual masks to represent different face regions, or other region segmentation techniques to model the discontinuities which may be difficult and tedious for users. Our method simplifies the preprocessing of deformable face models and can produce plausible results in both continuous and discontinuous regions on face models, as shown in Fig. 5. 46.

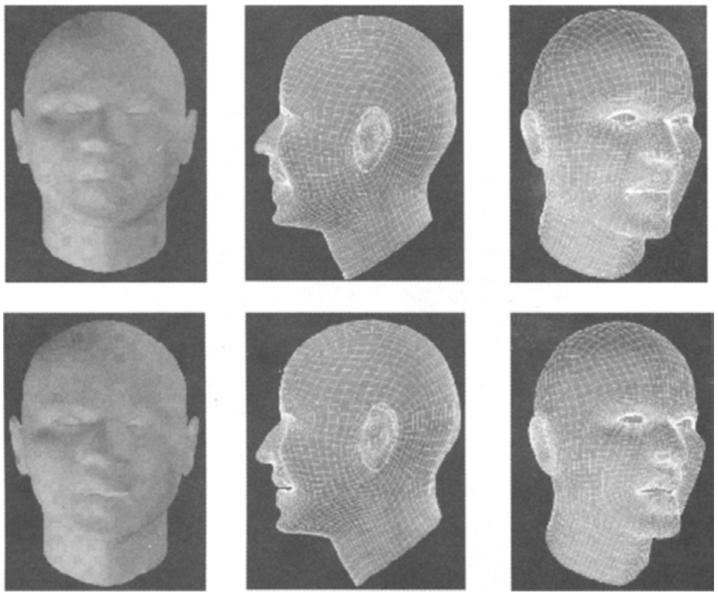


Fig. 5. 46 Generated anger expressions from facial motion data. Expression in the first row is generated via regression in Euclidean distance metric, where the lips (one of the most important discontinuities on human face) are stretched instead of opening mouth. The second row expression is from the same process except blending some Geodesic distance metric. The motions of lips are correctly modeled with mouth open

5.4.1.3 Blending of Euclidean and Geodesic Distances

A potential error may occur when the meshes contain only sparse triangles or quadrangles, in which cases the geodesic distances may be discrete and result in undesirable artifacts and distortions. We solve this problem by blending the geodesic and the Euclidean distances formulated as:

$$\delta_{ij} = w \cdot \delta_{ij} + (1-w) \cdot \gamma_{ij}$$

where $w \in [0, 1]$ is the blending coefficient. The Euclidean distance γ_{ij} works as a smoother part to eliminate the artifacts. The value of w can be decided empirically.

5.4.2 Bayesian Regression

While the geodesic manifold distance provides a tool for modeling the spatial structure of face models, the Bayesian regression can find the temporal features of facial animation. Although regression techniques like radial basis functions are enough for providing facial deformation results under noise-free circumstances, the processes of facial motion capture and 3D face scanning can hardly be noise-free. Bayesian approaches to regression [21-23] provide an elegant way of dealing with noise and uncertainty, as well as a framework of learning the statistical features of data. We use Bayesian approaches for exploring facial deformation driven by warping kernels, as shown in Fig. 5.47.

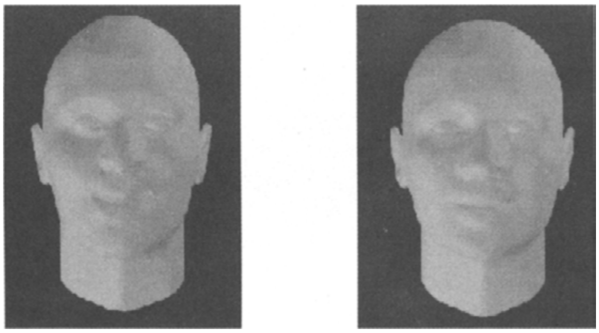


Fig. 5.47 Bayesian marginalization is applied to eliminate noise in motion or model data. The artifacts resulted from noisy data in (a) are smoothed in (b) in the generated expression

5.4.2.1 Framework of Bayesian Regression

In Bayesian interpretation of the regression problem, a nonlinear function $y(\boldsymbol{s})$ parameterized by the hyper-parameter $\boldsymbol{\theta}$ is assumed to underlie motion data $\{\boldsymbol{s}^{(n)}, t_n\}_{n=1}^{N}$ where N is the number of observations, or feature points. We denote coordinates of feature points at the neutral frame by $\boldsymbol{S}_N = \{\boldsymbol{s}^{(n)}\}_{n=1}^{N}$, and the correspondences at a target frame by three vectors $\boldsymbol{t}_N = \{t_n\}_{n=1}^{N}$ for respective warping channels of x, y or z. The inference of $y(\boldsymbol{s})$ by inferring the hyper-parameter $\boldsymbol{\theta}$ is depicted by the posterior probability distribution:

$$P(\boldsymbol{\theta}|\boldsymbol{t}_N,\boldsymbol{S}_N)=\frac{P(\boldsymbol{t}_N|\boldsymbol{\theta},\boldsymbol{S}_N)P(\boldsymbol{\theta})}{P(\boldsymbol{t}_N|\boldsymbol{S}_N)} \tag{5-33}$$

Minimizing the minus log of posterior probability distribution as an object function:

$$\boldsymbol{\theta}=\arg\min_{\boldsymbol{\theta}}(-\log\,(P(\boldsymbol{t}_N|\boldsymbol{\theta},\boldsymbol{S}_N)P(\boldsymbol{\theta}))) \tag{5-34}$$

Maximize a Posterior (MAP) and Maximum Likelihood (ML) are commonly used approaches for predicting outputs. However, they sometimes converge to local minima and are not convenient for generation of facial animations. Bayesian predictions can also be made without estimated value of $\boldsymbol{\theta}$ by marginalizing over the hyper-parameter. We perform integrations in equation (5-35) by sampling the possible values of $\boldsymbol{\theta}$ from $\boldsymbol{P}(\boldsymbol{\theta}|\boldsymbol{t}_N, S_N)$ using Markov chain Monte Carlo method of Hybrid Monte Carlo. The sampling process saves 200 sampled values of the hyper-parameter.

$$P(\boldsymbol{t}_{N+1} \mid \boldsymbol{t}_N,\boldsymbol{S}_N) = \int P(\boldsymbol{t}_{N+1} \mid \boldsymbol{\theta},\boldsymbol{S}_N)P(\boldsymbol{\theta} \mid \boldsymbol{t}_N,\boldsymbol{S}_N)\mathrm{d}\boldsymbol{\theta} \tag{5-35}$$

5.4.2.2 Modulation of Covariance Matrix with Geodesic Distances

There are many choices of covariance matrix under the condition of non-negative definite property for any set of points $\{\boldsymbol{s}^{(1)},\cdots,\boldsymbol{s}^{(n)}\}$. We empirically choose the following covariance matrix to adopt the geodesic distances.

$$\begin{aligned}\boldsymbol{C}(\boldsymbol{s}^{(i)},\, \boldsymbol{s}^{(j)}) &= v_0\exp\left\{-\frac{1}{2}\sum_{l=1}^{d}w_l(\boldsymbol{s}_l^{(i)}-\boldsymbol{s}_l^{(j)})^2\right\}+\\ &\quad a_0+a_1\sum_{l=1}^{d}\boldsymbol{s}_l^{(i)}\boldsymbol{s}_l^{(j)}+v_1\boldsymbol{\rho}(i,j)\end{aligned} \tag{5-36}$$

where $\boldsymbol{\theta}=\log\,(v_0, v_1, w_1, \cdots, w_n, a_0, a_1)$ represents the hyper-parameter since the variables in equation (5-36) are positive scale parameters. $\boldsymbol{\theta}$ corresponds closely to the hyper-parameter in neural networks to be adapted through learning.

The covariance function consists of three parts: the weighted sum term, the linear regression term (including a_0 and a_1) and the noise term. The first term expresses that the nearby inputs produce similar outputs and that the weights of w_l are usually set to ones. This leads to the equation below:

$$\begin{aligned}C(\boldsymbol{s}^{(i)},\boldsymbol{s}^{(j)}) &= v_0\exp\left\{-\frac{1}{2}\sum_{l=1}^{d}(\boldsymbol{s}_l^{(i)}-\boldsymbol{s}_l^{(j)})^2\right\}+\\ &\quad a_0+a_1\sum_{l=1}^{d}\boldsymbol{s}_l^{(i)}\boldsymbol{s}_l^{(j)}+v_1\boldsymbol{\rho}(i,j)\end{aligned} \tag{5-37}$$

where

$$\sigma_{ij} = \sum_{l=1}^{d} (s_l^{(i)} - s_l^{(j)})^2 \tag{5-38}$$

is the squared Euclidean distance.

We substitute the geodesic distance $\delta_{ij} = dijk(s^{(i)} - s^{(j)})$ for equation (5-38). and get:

$$C(s^{(i)}, s^{(j)}) = v_0 \exp\left\{-\frac{1}{2} dijk(s^{(i)} - s^{(j)})^2\right\} + a_0 + a_1 \sum_{l=1}^{d} s_l^{(i)} s_l^{(j)} + v_1 \boldsymbol{p}(i, j) \tag{5-39}$$

With the covariance matrix $\boldsymbol{C}$, equation (5-34) can be deduced for N training pairs as:

$$\boldsymbol{\theta} = \arg\min_{\theta}\left(\frac{1}{2}\log\det(\boldsymbol{C}) + \frac{1}{2}\boldsymbol{t}_N^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{t}_N + \frac{N}{2}\log 2\pi\right) \tag{5-40}$$

5.4.3 Results and Discussions

The data used in this section are captured with Motion Analysis motion capture system. Facial motion data is captured with a set of 42 markers (including 3 markers on a head mounted jig for global motion estimation) at a frame rate of 60 fps. Eight representative expressions are extracted, as shown in Fig. 5.48, and used to drive an ordinary quadrangle face mesh via manifold Bayesian regression, as in Fig. 5.49. The results show that Bayesian method for regression and the geodesic distances are suitable for representing the underlying deformation paradigm and the complex topology of human faces. As shown in Fig. 5.49, the generated expressions are smooth and the correct movement of discontinuities in the mouth region

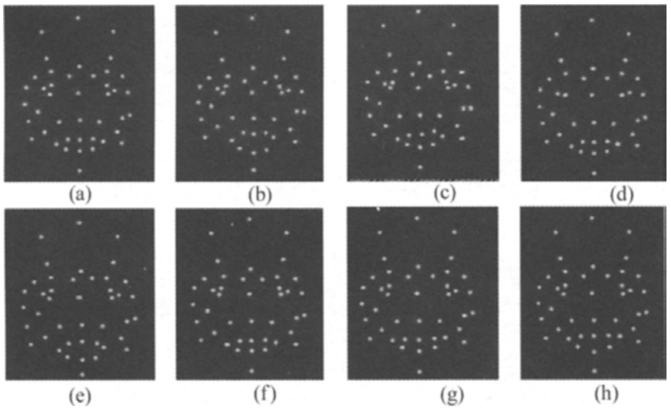


Fig. 5.48 Eight key expressions extracted from facial motion data. (a) Neutral; (b) Anger; (c) Disgust; (d) Eye-close; (e) Fear; (f) Sad; (g) Smile with mouth closed; (h) Smile with mouth open

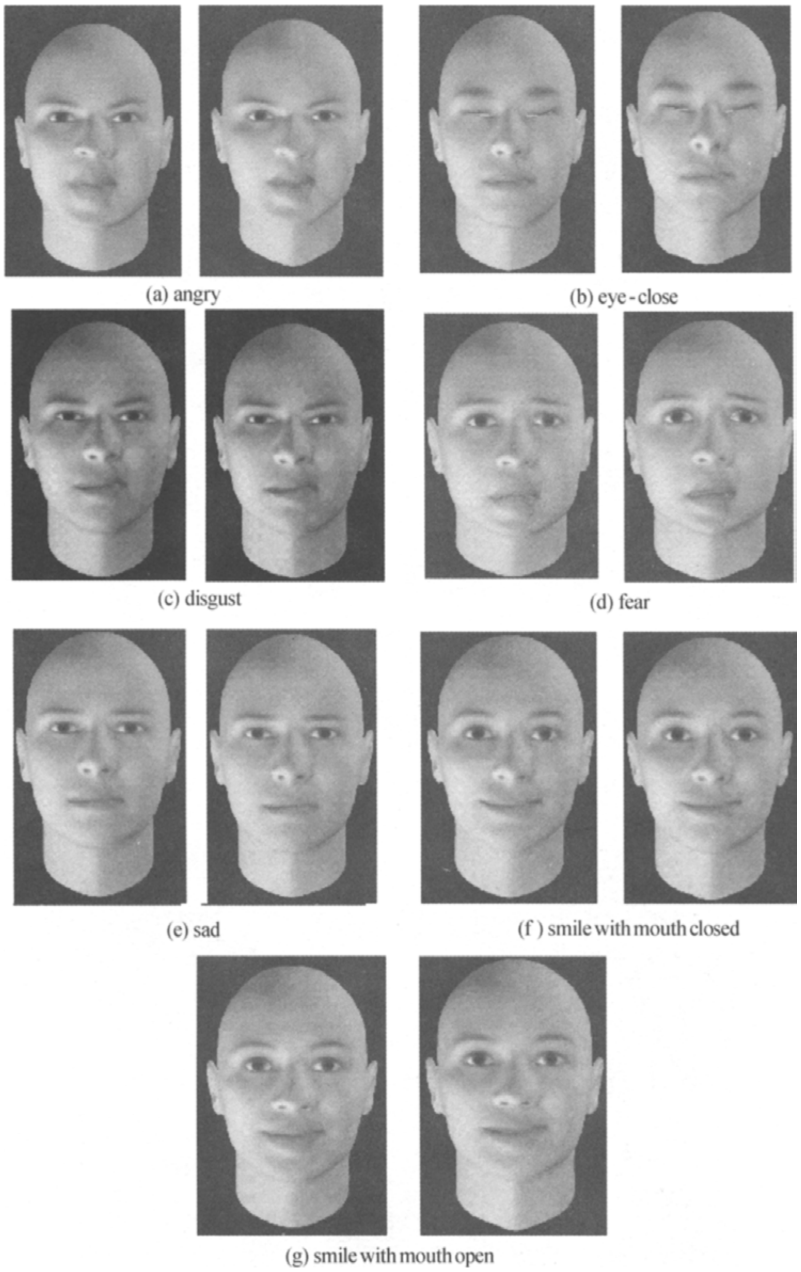


Fig. 5.49 The comparison of the results by both Euclidean and Bayesian regression for different expressions. The left ones of pairs are generated in the Euclidean distance metric only and the right ones are in the mixture metric of geodesic distances and Euclidean distances

shows the power of manifold. The situation in the eyes' regions may be confusing as they show hardly any difference between the left and the right expressions. The reason is that they differ from the mouth region in the neutral expression as they have proper spatial gaps between the upper and the lower eyelids by which the Euclidean distances can equally well express the discontinuities.

In this section, we present a novel and effective method for driving facial animations based on motion data. Geodesic manifold distances are adopted in the framework of Bayesian learning to automatically model the discontinuities and complex topology of human faces. After the covariance matrix is properly modulated with the geodesic distances, Hybrid Monte Carlo method is then used to compute the integral of probabilities to predict results under noisy circumstances. The techniques in this section allow facial motion data to be applied to any facial mesh. Previously, adapting facial motion data to an individual mesh required much more artistic intervention.

Techniques in this section can be used in many scenarios, such as human computer interaction and digital entertainment. They also have prospective applications in data compression for 3D streaming media, or distant meeting for facial communication between participants as the computing power grows.

References

1. Liu Z, Shan Y, Zhang Z. Expressive expression mapping with ratio images. In: SIGGRAPH'01, pp. 271-276, ACM Press, 2001.
2. Noh J. Neumann U expression cloning. In: SIGGRAPH'01, pp. 277-288, ACM Press, 2001.
3. Zhang Q, Liu Z, Guo B, Shum H. Geometry-driven photorealistic facial expression synthesis. In: SCA'03, pp. 177-186, ACM Press, 2003.
4. Pyun H, Kim Y, Chae W, Kang HW, Shin SY. An example-based approach for facial expression cloning. In: SCA'03, pp. 167-176, ACM Press, 2003.
5. Tenenbaum JB, Silva V, Langford JC. A global geometric framework for nonlinear dimensionality reduction. Science, 290(5500): 2319-2323, 2000.
6. Freeman WT, Jones TR, Pasztor EC. Example-based super-resolution. IEEE Computer Graphics and Applications, 22(2): 56-65, 2002.
7. Boykov Y, Veksler O, Zabih R. Fast approximate energy minimization via graph cuts. In: ICCV'99, pp. 377-384, IEEE Computer Society, 1999.

8. Martinez AM, Benavente R. The AR face database. CVC Technical Report 24, 1998.
9. Roweis ST, Saul LK. Nonlinear dimensionality reduction by locally linear embedding. Science, 290(5500): 2323-2326, 2000.
10. Cootes TF, Taylor CJ, Cooper DH, Graham J. Active shape models-their training and application. Computer Vision and Image Understanding, 61(1): 38-59, 1995.
11. Isard M, Blake A. Contour tracking by stochastic propagation of conditional density. ECCV'96, pp. 343-356, Springer, 1996.
12. Tomasi C, Kanade T. Shape and motion from image streams under orthography: a factorization method. International Journal of Computer Vision, 9(2): 137-154, 1992.
13. Xiao J, Chai JX, Kanade T. A closed-form solution to non-rigid shape and motion recovery. In: ECCV'04, pp. 573-587, Springer-Verlag, 2004.
14. Blanz V, Vetter T. A morphable model for the synthesis of 3D faces. SIGGRAPH'99, pp. 187-194, ACM Press, 1999.
15. Pighin FH, Hecker J, Lischinski D, Szeliski R, Salesin D. Synthesizing realistic facial expressions from photographics. SIGGRAPH' 98, pp. 75-84, ACM Press, 1998.
16. Lee S, Wolberg G, Shin SY. Scattered data interpolation with multilevel B-splines. IEEE Transactions on Visualization and Computer Graphics 3(3): 228-244, 1997.
17. Noh JY , Fidaleo D, Neumann U. Animated deformations with radial basis functions. VRST 2000, pp. 166-174, 2000.
18. Blanz V, Basso C, Poggio T, Vetter T. Reanimating faces in images and video. Computer Graphics Forum (EuroGraphics'03), 22(3): 641-650, 2003.
19. Williams L. Performance-driven facial animation. SIGGRAPH'90, pp. 23 - 242,ACM Press,1990.
20. Niyogi P, Belkin M. Laplacian eigenmaps for dimensionality reduction and data representation. Technical Report 2002-01, University of Chicago, 2002.
21. Hertzmann A. Machine learning for computer graphics: a manifesto and tutorial. In: Proceeding of 11th Pacific Graphics, pp. 22-36, IEEE Computer Society, 2003.
22. MacKay DJC. Introduction to Gaussian processes. In: Bishop CM, ed. , Neural Networks and Machine Learning. pp. 133-166, Kluwer Academic Press, 1998.
23. Williams CKI, Rasmussen CE. Gaussian processes for regression. In: NIPS 8, pp. 514-520, MIT Press, 1996.

6

Intelligent Techniques for Processing and Management of Motion Data

Motion Capture (MoCap) systems are widely used in computer animations, simulations and video games. Recently, a large commercial MoCap database is available and it will be of great importance to the reusability of motion data as well as the efficiency of computer animations.

Several key techniques in the MoCap database system include: (1) motion data prediction such as data format conversion, metadata segmentation and extraction, etc; (2) motion data abstraction, such as key-frame extraction, index building, keyword annotating; (3) motion data retrieval.

But for the time being, some problems still exist in the construction and operation of the database which we cannot overlook. For example, too much human interaction is involved thus causing low efficiency in its intelligent functions. All these drawbacks are embodied mainly in three aspects as below. First, operators need to do the segmentation and extraction work of motion data by hand during the motion data preprocessing stage, which will cause low efficiency. Second, key-frame extraction by hand for motion data has its difficulty and inaccuracy to a certain extent. Meanwhile, data retrieval is based on keyword annotating, and such labor-intensive and subjective job would cause the inaccuracy of the results. Third, data retrieval is dependent on keyword-based data retrieval which demands annotating every motion data sequence by hand and this is absolutely time-consuming and tedious work.

Basically speaking, 3D motion data can be considered one kind of multimedia data. After so many years' study on intelligent analysis and management of multimedia data, we have proposed several techniques concerning the MoCap database, specifically for automatic segmentation of 3D human motion data into primitive actions, key-frame extraction from motion capture data and content-based motion capture data retrieval technique. We will describe in detail the above techniques in the following sections.

6. 1 Automatic Segmentation of 3D Human Motion Data into Primitive Actions

Automatic segmentation of 3D human motion data into primitive actions means segmenting long human motion sequence into divided primitive actions (such as walking, jumping, kicking and punching) so that each motion clip has its individual semantic meaning. These divided clips are then stored into MoCap database so as to be conveniently manipulated in animation authoring systems or be retrieved based on keyword or visual content. Why not do we just capture individual primitive actions and save the data separately to simply avoid segmentation? The reasons are as follows:

- Long motion sequences are more comfortable for actors to perform and contain more natural transitions from one action to the next. To get a more natural motion over a long time, the only choice is to capture a long motion sequence.
- Capturing a long motion sequence is more efficient than capturing short motion clips one by one.

6. 1. 1 Overview of Motion Data Segmentation

Motion data segmentation has always been the hot research area. Existing motion (activity) segmentation or recognition methods can be categorized into two fields: (1) segmenting or recognizing different human activities in video streams [1,2]; (2) segmenting or recognizing distinct human activities in 3D motion data. In video streams, motion segmentation and recognition are mainly used for human activities analysis and surveillance. And for 3D motion data, motion segmentation and recognition are mainly used for animation production. In this chapter, our goal is to segment long 3D human motion sequences into divided primitive actions, so here we only focus on the related work that shares this goal.

Arikan, et al. [3] proposed a model-based approach to motion annotation. In order to annotate all motion data in database with different description words, such as "running" and "walking", they built a Singular Value Decomposition (SVM) classifier. Based on the SVM classifier and some hand-annotated training samples, a large set of motions could be annotated automatically. And also this method could be used to segment motion data. Kahol, et al. [4,5] proposed an algorithm called Hierarchical Activity Segmentation. This algorithm employs a dynamic hierarchical layered structure to represent the human anatomy and uses low-level motion parameters to characterize motion in the various layers of this hierarchy. Then a naive Bayesian classifier was applied to learn the criteria for gesture segmentation that is estimated from empirical data provided by experts. However, the performances of these approaches heavily rely on the annotated training samples or empirical data from experts. Lu, et al.[6] presented a two-threshold, multidimensional segmentation algorithm to automatically decompose a complex motion into a sequence of simple linear dynamic models. But this algorithm was developed for repetitive motion instead of general and non-repetitive human motion.

Many researches proposed that low-level motion segmentation could be derived by some straightforward methods. Fod, et al.[7] segmented human arm movement data into primitives by detecting zero crossing of angular velocities. Pomplun and Mataric[8] used the same idea, in which they described a joint-space based segmentation and comparison algorithm. The limitation of these approaches is that a longer motion sequence will be divided into more sections than the actual number of primitive actions. Wang, et al.[9] explored a more sophisticated technique which segmented motion sequences into atomic components and clustered them together using a Hidden Markov model.

Barbic, et al.[10] assumed that different simple human motions should have different intrinsic dimensionality, so they assigned a cut when the intrinsic dimensionality of the motion's local model suddenly increase based on Principal Component Analysis (PCA) technique. Further they proposed a Probabilistic PCA (PCA) method for segmentation based on the assumption that the Gaussian models of two separate behaviors are quite different. These two methods work well except that the computation efficiency is a little low. For each new incoming frame, system runs the PCA or PPCA algorithm again. And in [10], they proposed a Gaussian Mixture Model (GMM) segmentation method based on the assumption that the frames from different simple motions form separate clusters, and each cluster can be described reasonably well by a Gaussian distribution. But in this method, the number of clusters should be set by user in advance, which is not practical for segmenting a large motion database where many motion sequences with different number of behaviors are included.

All the methods above have their own shortcomings. Now in the following section a very simple and easy way to implement segmentation technique will be presented, which can achieve higher precision and efficiency. A nonlinear dimensionality reduction technique is used to map original motion sequences into low-dimensional manifold, and then clustering techniques are applied to segment primitive actions apart.

6.1.2 An Automatic 3D Human Motion Data Segmentation Approach Based on Nonlinear Dimensionality Reduction

6.1.2.1 Motion Data Preprocessing

Human body is modeled by an articulated skeleton with some rigid limbs which has the same topological structure as human body. In this skeleton, it is defined that the body contains 16 joints that are constructed by a tree structure. Joint *root* is the root of the tree and those paths from root to all endmost joints in human skeletal model form sub-trees of root. Joint root is represented by a 3D translation vector and a 3D rotation vector. The translation of joint root determines the current position of the skeleton while the rotation of joint root determines the overall orientation of the skeleton. For the other joints, rotation describes their orientation in the local coordinates of their parent joints. The factors all above determine the posture of human body. A motion sequence $\boldsymbol{M}$ with n frames is presented as

$$\boldsymbol{M}=\{\boldsymbol{F}(1),\boldsymbol{F}(2),\cdots,\boldsymbol{F}(t),\cdots,\boldsymbol{F}(n)\} \tag{6-1}$$

$$\boldsymbol{F}(t)=\{\boldsymbol{p}(t),\boldsymbol{q}_1(t),\cdots,\boldsymbol{q}_m(t)\} \tag{6-2}$$

where $\boldsymbol{F}(t)$ is the tth frame in motion sequence $\boldsymbol{M}$, $\boldsymbol{p}(t)$ is the translation of the root joint, $\boldsymbol{q}_i(t)$ is the rotation of joint i in frame t, m is the number of joints used in human skeleton.

All of the motions are performed by a real actor and recorded by an optical motion capture system at frame rate 120 fps. Each motion $\boldsymbol{M}$ is presented by the same skeleton with 51 DOFs (corresponding to 16 joints of human body, see Fig. 6. 1). Before inputting motion sequence $\boldsymbol{M}$ into segmentor, we filter out the translation and rotation of root joint which present the overall position and orientation of human skeleton and thus have no relation with specific primitive action. So in original motion data space, each frame of motion sequence is represented as a vector of 45 dimensions.

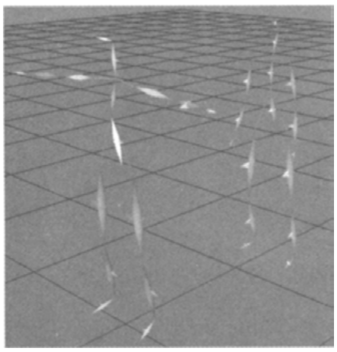


Fig. 6. 1 The skeleton model

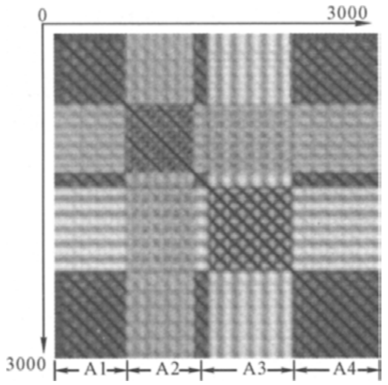


Fig. 6. 2 Distance matrix of motion

6.1.2.2 Coarse Segmentation

Fig. 6. 2 gives the distance matrix of one motion sequence which represents the distances between each frame pairs in this motion sequence. This motion sequence has about 3,000 frames and contains 4 primitive actions. As shown in Fig. 6. 2, we can intuitively judge that this motion sequence contains 4 or 5 different primitive actions because there are some clear borderlines.

Life would be easier if the edge detection methods could see these borderlines clearly. However, there is so much noise in distance matrix that these borderlines cannot be detected accurately by most of the edge detection methods. In order to simplify this problem, we just use the distance curve of the first frame in motion sequence. The distance curve of specific frame represents the temporal distance sequences between this frame and all rest frames in the same motion sequence, which can be considered as a slice of distance matrix (see Fig. 6. 3).

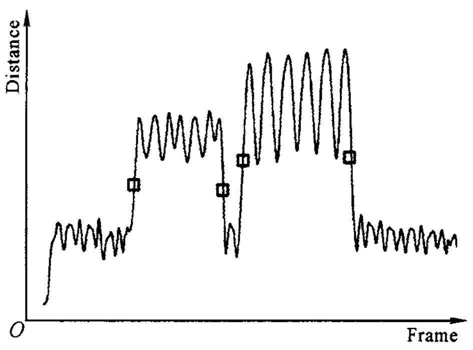


Fig. 6. 3 Distance curve (the lst frame). This figure shows the distances between the lst frame and all rest frames in the same motion sequence. The box represents the candidates of coarse segmentation point

If we consider the distance curve in Fig. 6. 3 as digital signals, we can detect the coarse segmentation point at the positions where signals change suddenly (see the boxes in Fig. 6. 3). Following is the pseudocode of our heuristic method of coarse segmentation.

```
M = LoadMotion (··· );              // load motion sequence from MoCap database
D = DistanceMatrix ( D );           // calculate distance matrix of motion sequence
len = Length ( M );                 // get the length of this motion sequence
curFrm =1;                          // set current frame number to first frame
while ( curFrm < len )
{
    curDistCurv = D ( curFrm, : );
                                    // pick out the curFrm-th distance curve from D
    interval = 120;                 // set number of frames to be processed for each
                                       iterative step
                                    // calculate the max-min difference for the current
                                       motion ( from curFrm+60 to curFrm+240)
    diff = MaxMinDiff ( curDistCurv ( curFrm+sp : curFrm+ep ) );
    i = curFrm + ep;
    while ( i < len )
    {
        [ maxvalue maxindex ] = max ( curDistCurv ( i : i + interval ) );
        [ minvalue minindex ] = min ( curDistCurv ( i : i + interval ) );
        if ( maxvalue - minvalue ) >= diff * α
        {
            segPt = i + fix ( ( maxindex + minindex ) / 2 );
            break;
        }
```

```
            if ( maxvalue - minvalue ) <= diff * β
            {
                segPt = i + fix ( ( maxindex + minindex ) / 2 );
                break;
            }
            i = i + interval;
        }
        Save ( segPt );
        curFrm = segPt;
    }
```

For each new start point *curFrm*, it firstly reads in the distance curve of *curFrm* between frame *curFrm*+*sp* and frame *curFrm*+*ep* to calculate the max-min difference value *diff* which we consider as the distance changing range of current action. Usually we set $sp=60$ and $ep=240$ and this initial calculation means that each action to be segmented should last at least 2 seconds for the algorithm to detect it. Then at each iterative step, the algorithm reads in the following *interval* (we use 120 frames in experiments) frames of distance curve and calculates the max-min difference value of them. If the new difference value is much larger (or smaller) than *diff* (α and β are used to control these thresholds, in experiments we set $\alpha=0.75$ and $\beta=1.25$), a new coarse segmentation point should be placed at the middle between the max value and the min value. For a candidate set of coarse segmentation points, if there are two points whose distance is smaller than 240 frames, we should merge them to one point because we assume that each action to be segmented should last at least 2 seconds (240 frames).

Now we can get the coarse segmentation point between every two primitive actions in the motion sequence by the heuristic segmentation method above. But unfortunately, the coarse segmentation points are not accurate enough for primitive data segmentation. In the next section, a method combining ISOMAP and K-mean clustering will be described which can segment the motion sequence into primitive actions accurately.

6.1.2.3 Accurate Segmentation Based on Nonlinear Dimensionality Reduction and K-mean clustering

Human motion data resides in high-dimensional space. For example, a motion clip with 30 frames and 51 DOFs is represented as a vector with 1,530 dimensions. This leads to two problems: (1) high-dimensional data results in more computation time; (2) high-dimensional data have complex structure and is more difficult to analyze. So dimensionality reduction should be brought in here to get high efficiency and precision.

The classical techniques of dimensionality reduction, such as PCA and Multi-dimensional Scaling (MDS), are easy approaches to implement, efficiently computable and guarantee to discover the true structure of data lying on or near a linear subspace. PCA finds a low-dimensional embedding of the data points that best preserves their variance as measured in the high-dimensional input space. Classical MDS finds an embedding that preserves the interpoint distançes, equivalent to PCA when those distances are Euclidean. But unfortunately, human motion data are very complex and have nonlinear structures that are invisible to PCA or MDS. So if PCA or MDS is applied to reduce the dimensionality of human motion data, much useful information will be discarded.

Here we use ISOMAP [11] to do this challenging job, which combines the major algorithm features of PCA and MDS—computational efficiency, global optimization, and asymptotic convergence guarantees, with the flexibility to learn a broad class of nonlinear manifolds. ISOMAP can not only reduce the dimensionality of high-dimensional input space, but also find meaningful low-dimensional structures hidden behind their high-dimensional observations (See Fig. 6. 4, classical "Swiss roll" illustration of ISOMAP). For the detailed ISOMAP algorithm please refer to [11].

As we saw earlier, motion data sequences in original space lie on high-dimensional, twisted and folded manifold, which makes it very difficult to cluster similar poses together to form primitive actions. But after nonlinear dimensionality reduction, segmentor can easily distinguish dissimilar poses by some simple clustering techniques, such as K-mean [12] clustering technique.

In Fig. 6. 5, a motion sequence with about 600 frames illustrates the power of ISOMAP. This motion sequence contains two primitive actions (normal walking motion firstly and then followed by a side-walking motion), and there is a natural transition between them. After nonlinear re-

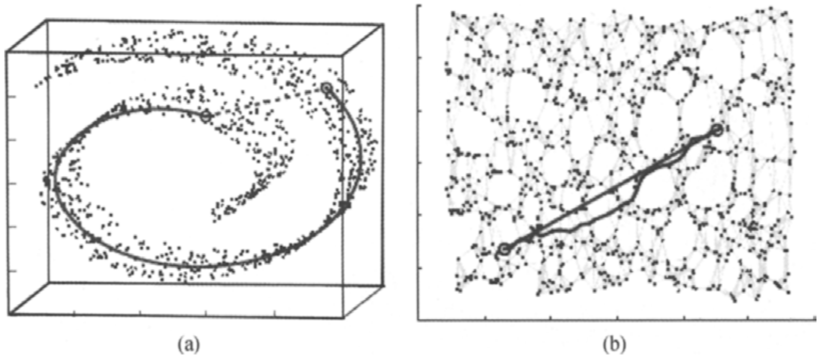


Fig. 6. 4 The "Swiss roll" data set [11]. (a) For two points on a nonlinear manifold, their Euclidean distance in the high-dimensional input space may not accurately reflect their intrinsic similarity, as measured by geodesic distance along the low-dimensional manifold; (b) The two-dimensional embedding recovered by ISOMAP. Straight line in the embedding represents simpler and clean approximations to the true geodesic paths than do the corresponding graph paths

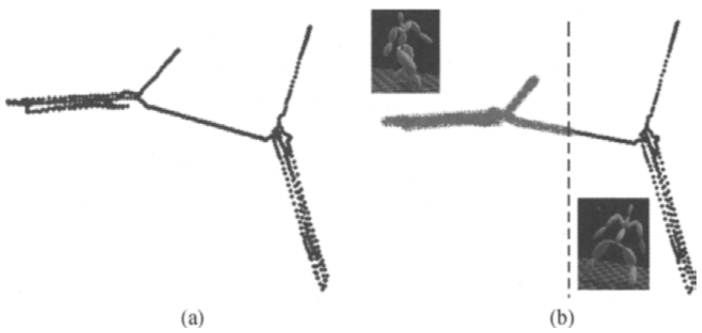


Fig. 6. 5 Motion sequence in low-dimensional (3D) manifold. (a) Low-dimensional embedding of original motion sequence after ISOMAP; (b) K-mean clustering technique is applied to low-dimensional embedding. The left "stars" correspond to the normal walking motion in original motion data space, and the right "dots" correspond to the side-walking motion in original motion data space

duction by ISOMAP (see Fig. 6. 5(a)), we can get the low-dimensional embedding of original motion sequence with a very simple structure. Then K-mean clustering technique is applied to cluster the similar points in low-dimensional embedding together to form two primitive actions (see Fig. 6. 5(b)). After clustering we can see that the straight line (consists of some points) between two point clouds represents the transition between two primitive actions and the segmentation point is found in the middle of this line, which is very intuitive.

Fig. 6. 6 gives another more complex example. This motion sequence contains two very similar primitive actions and there is a natural transition between them. Picture (a, b, c) is a knocking-floor motion (4 repeated cycles) and picture (d, e, f) is the following brushing-floor motion (5 re-

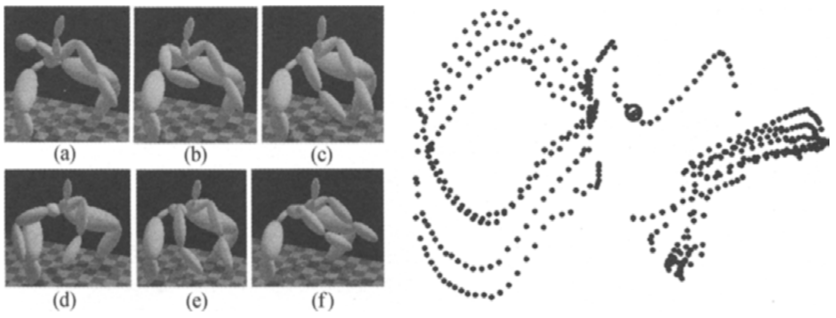


Fig. 6. 6 (Left) Tow primitive actions with natural transition. (a, b, c) is a knocking-floor motion, (d, e, f) is a brushing-floor motion; (Right) Low-dimensional (3D) embedding of original motion sequence after ISOMAP. The circled dot is the segmentation point found by K-mean clustering

peated cycles). As shown in the figure, the pose in picture (c) is very similar to that in picture (e), but actually they belong to different primitive actions and occur at different time. This observation means that if we cluster all of the poses (frames in motion sequence) in original motion data space directly, it will be very difficult to get the right segmentation point. Clustering results in Fig. 6. 7 prove that our supposition is right. The left segmentation is the K-mean clustering result from low-dimensional embedding of original motion sequence after ISOMAP, and the right segmentation point can be detected easily at the time of signal changing (see Fig. 6. 7(a)). But given the K-mean clustering result from original motion sequence without ISOMAP, the right segmentation point cannot be detected (see Fig. 6. 7(b)). The valleys of signals are the frames around the pose in picture (e) in Fig. 6. 6, which are clustered to knocking-floor motion.

Having found the coarse segmentation point between every two primitive actions in the previous section, we can deliver each pair of primitive actions to the above method to get accurate segmentation point.

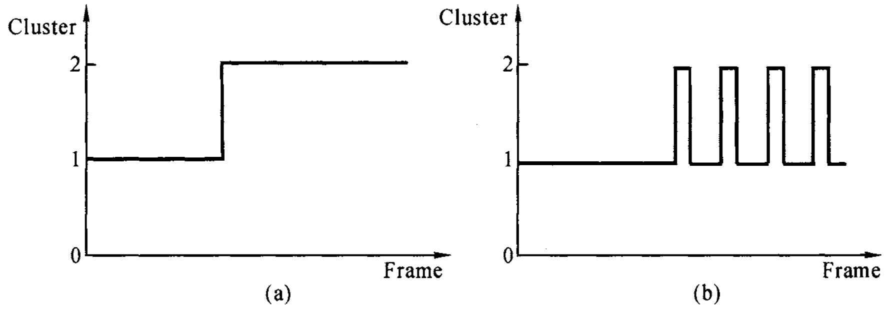


Fig. 6. 7 Results from K-mean clustering. (a) K-mean clustering result from low-dimensional embedding of original motion sequence after ISOMAP; (b) K-mean clustering result from original motion sequence without ISOMAP

6. 1. 3 Results and Discussions

A prototype system by Matlab is developed to test this method, which runs on a PC with PentiumIV 2. 8 GHz CPU and 1 G memory. And a MoCap database with 20 motion sequences (72,055 frames) is used in this experiment. Each sequence includes at least 2 or more actions. Most of the typical human actions are performed by actors and stored in these motion sequences, such as walking, running, kicking, punching, jumping, cleaning window, washing floor, and sweeping floor. In these experiments, we also develop a prototype system with PPCA method[10] by C++ and compare the performance of our method with those of PPCA method and transitions selected by human manually. Fig. 6. 8 shows some experimental results. Figs. 6. 9 and 6. 10 show key poses corresponding to each primitive action of one of the motion sequences in MoCap database.

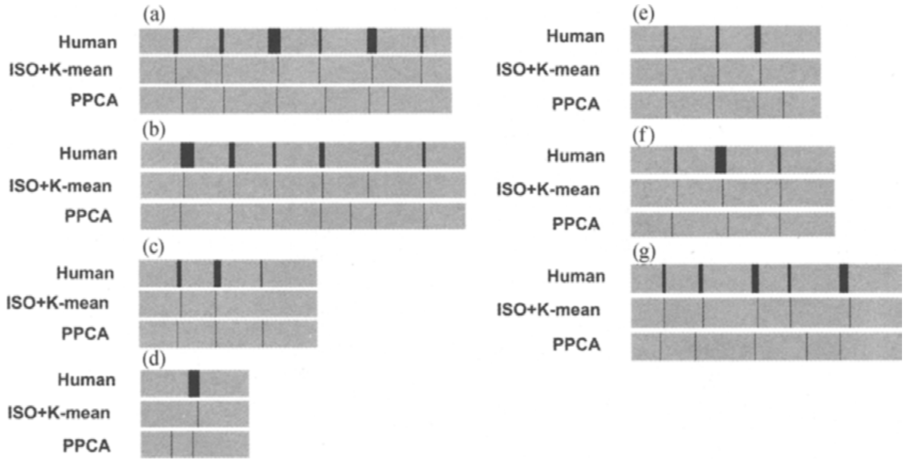


Fig. 6. 8 Some segmentation results from our experiments, comparing with ground-truth from human manual segmentation and PPCA method. Each figure corresponds to one motion sequences from the MoCap database. Human gives a ground-truth of segmentation for every motion sequence, which is a motion transition range. All segmentation points falling into these ranges are acceptable

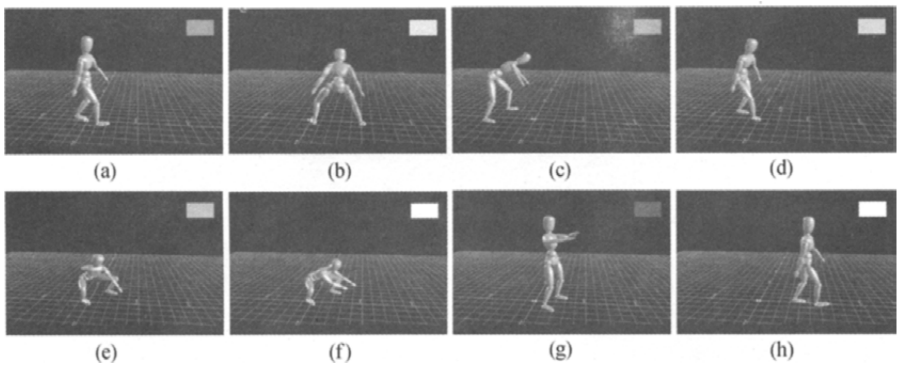


Fig. 6. 9 Key poses correspond to primitive actions of one of motion sequences from MoCap database. Primitive actions (a) to (h) are: walking, side-walking, sweeping floor, walking, knocking floor, washing floor, cleaning window, and walking

Table 6. 1 compares the performance of our method with those of PPCA and human manual segmentation. As shown in the table, our method gets higher precision than PPCA while PPCA gets higher recall than our method. There are two reasons for this phenomenon.

Firstly, PPCA method usually detects more incorrect segmentation points than our method (see Fig. 6. 8 (b), (d) and (e)), which results in lower precision. The PPCA method is based on the assumption that Gau-

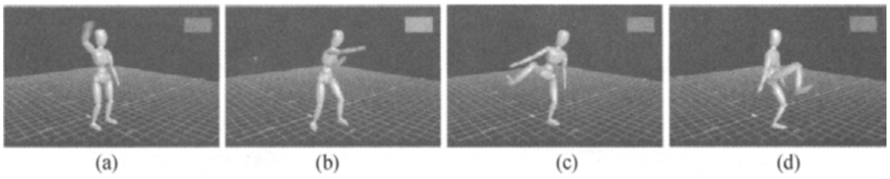


Fig. 6. 10 Segmentation results of motion sequence. Primitive actions (a) to (d) are: waving, boxing, side-kicking, and forward-kicking

Table 6. 1 Comparison of performance of our method with those of PPCA and human manual segmentation

Method	Precision	Recall	Average Time
ISO+K-mean	93. 9%	92. 5%	4 s/clip (Matlab)
PPCA	91. 5%	93. 2%	10 s/clip (C++)
Human	—	—	12 s/clip

ssian models of two distinct actions should be quite different, and the intrinsic dimensionality of a motion sequence containing a single action should be smaller than that containing multiple actions. PPCA models the variation of motion data distribution linearly and is sensitive to the variation of data distribution. So if the range of the same action becomes larger or smaller than anterior action data, PPCA will give us a new segmentation point. But our coarse segmentation algorithm calculates the Euclidian distance between different poses directly, which can hide the reasonable range changing of the same type of action by setting appropriate threshold parameter.

Secondly, our method occasionally misses some correct segmentation points (see Fig. 6. 8 (c) (d)), which results in a lower recall. It assumes that original motion data reside in a high-dimensional manifold, which can be mapped into a low-dimensional embedding and classified by normal clustering techniques. But for some action pairs which have many very similar poses, this method will not work, because these similar poses from different actions will assemble in the same local region of low-dimensional embedding. As Fig. 6. 11 shows, two kicking actions, side-kicking and forward-kicking, are mapped into corresponding low-dimensional embedding. These two actions both start and end with standing pose, and all the standing poses are mixed in the low-dimensional embedding (see the rectangle of Fig. 6. 11). So this method cannot accurately segment these two kicking actions apart.

Table 6. 1 shows that the efficiency of our method is much higher than PPCA method and human manual segmentation. In PPCA method, for each new incoming frame the system will run the whole PPCA segmentation algorithm again and judge whether there is a segmentation point. But in our method, we just compute distance curve for each interval with 120

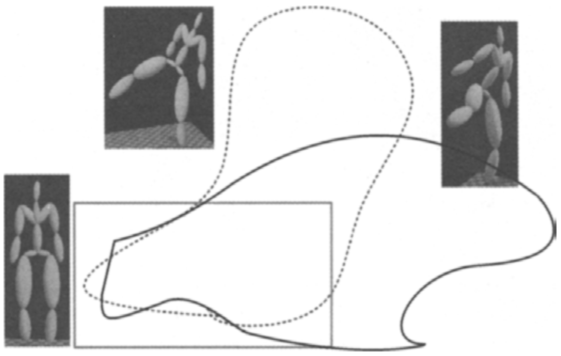


Fig. 6. 11 Low-dimensional (3D) embedding of two kicking actions. Solid cycle is a side-kicking motion, dotted cycle is a forward-kicking motion. Points in box are standing poses

frames and judge whether there is a coarse segmentation point. If so, ISOMAP and K-mean clustering algorithm will be applied to find fine segmentation point. Suppose that there is an original motion sequence with N frames and M distinct actions, PPCA method will run the whole PPCA segmentation algorithm (including standard PCA subspace construction, Gaussian model construction, computing Mahalanobis distance, etc.) $(N-M\times 240)$ times (240 is the initial number of frames for PCA subspace construction). For our method, we only run coarse segmentation algorithm about $((N-M\times 240)/120)$ times (240 is the initial number of frames for computing the distance changing range of current action and 120 is the value of interval parameter for coarse segmentation point detection), and run ISOMAP and K-mean clustering algorithm M times.

Although our method has high efficiency, it has drawbacks and some limitations. For action pairs which have many very similar poses, our method will miss the segmentation points. One potential solution for this problem is to take temporal information into account when clustering technique is applied, which means that temporal adjacent poses (frames) should have more probability to belong to the same action. The similar poses which lie far away from each other on temporal axis should not be classified into the same action.

And if there is a very long motion sequence in which user just wants to use several action clips, it's not necessary to segment it into many distinct actions wholly. We just hope that the system can pick out the desired action clips for user. So in future, we will add more user interactive operation into our method which means when users browse a very long motion sequence, they can select one or more frames interactively and then the system will pick out the distinct action clips which contain these frames automatically.

6.2 Motion Data Abstraction

With the rapid development of motion capture system, motion capture data can be easily acquired. However, the high dimension, loose structure of motion data prevent it from efficient application. The high efficiency in motion data management is the key to solve the problems above.

Motion data abstraction means extracting the low-dimensional features of original 3D motion data for the use of database indexing. This idea stems from that in content-based multimedia retrieval. According to present research advances, most of the work is concentrated on motion data key-frame extraction, which means extracting key-frame postures (such as extreme postures) that can well describe the current motion clip from raw motion data sequences, similar to the approach of video key-frame extraction. In this way, the retrieved key-frames are the abstraction of raw motion data sequences.

6.2.1 Overview of Motion Key-frame Extraction

At present, massive motion data have been acquired by MoCap systems and these data is extensively used in different areas such as computer game, computer animation and medical simulation. Motion data are captured in high sampling frequency, so key-frame (key posture) extraction is important for its compressing, storage, retrieval, browsing and motion editing.

Key-frame extraction has been extensively explored recently in multimedia information processing where key-frames are used in video browsing and content-based video retrieval applications [13-15].

However, in the field of motion capture, key-frames describe three-dimensional skeleton data rather than two-dimensional temporal data in video analysis field. At present, motion data key-frame extraction techniques fall into two categories: uniform sampling and adaptive sampling.

The simplest idea for key-frame extraction from motion data sequence is uniform sampling, which extracts key-frames with uniform sampling intervals. This method is simple and costs less time, but it cannot summarize the original human motion effectively because of over-sampling in segments with less pose changes or under-sampling in segments with great pose changes. Over-sampling results in data redundancy and under-sampling leads to loss of motion information.

Adaptive sampling methods can tackle this problem. Adaptive sampling extracts key-frames according to performer's pose changes rather than motion time. These methods result in less key-frames in motion segments with less pose changes and more key-frames in motion segments with great pose changes so that they summarize original human motion more effectively than the uniform sample method. To use this method, an error tolerance should be specified as the distance criterion. This method summarizes original motion effectively, but it extracts key-frames regardless of human motion's geometric meaning so that the extracted key-frame collection cannot guarantee the consistency between similar motions. In addition, error parameter should be specified manually before using those methods to extract key-frames. Some researchers used clustering techniques to extract key-frames. Liu, et al. [16] proposed a clustering-based method to extract key-frames adaptively. In this method, a similarity measurement between two frames is defined. Using the defined similarity, N frames of motion data are classified into K clusters and the first frames of each cluster are considered key-frames. In clustering, a specified error threshold determines the number of cluster collections (key-frames). Shen, et al. [17] proposed a frame distance which was defined as the total amount of rotation of all articulations on human body and chose key-frames by means of comparing frame distance of every two adjacent frames. The method proposed by Lim and Thalmann [18] also falls into this category. They treated motion data as high-dimensional curves, and then applied a simple curve simplification algorithm to extract key-frames. There are two major aspects we need to pay attention to for these adaptive the key-frame extraction methods. One is how the motion's physical features can be efficiently represented, and the other is how to quantitatively analyze key-frame extraction methods.

In order to deal with motion data, several motion feature representations have been proposed. Liu, et al. [16] applied a hierarchical motion representation. All joints in human skeleton are divided into 5 layers and parent joint is more prominent than child joint. Lee, et al. [19] described a two-layer structure for representing human motion data. The lower layer is a Markov process that retains the details of the original motion data, while the higher layer is a statistical model that provides support for the user interfaces by clustering the data to capture similarities among character states. The two representations generalize motion data, but motion's physical features cannot be represented clearly. Chui, et al. [20] proposed local spherical coordinates relative to the root orientation as the segment posture of each skeletal segment. This representation achieves affine invariance of body transformations and decreases the dimension of motion feature. However, every skeletal segment is represented by two parameters which cannot benefit to observe posture of each skeletal segment. Mueller, et al. [21] introduced 31 Boolean features expressing geometric relations between certain body points of a pose. This method can represent motion's physical feature very well, but the number of features is too large.

In order to quantitatively analyze key-frames, some researchers proposed quantitative indices such as error rate and compression ratio. As mentioned above, Liu, et al. used a threshold to determine whether a frame belongs to a certain cluster; Shen, et al. computed the distance between current frame and the last frame in the key-frame collection and then compared this distance with a certain threshold to judge whether it belongs to key-frame; Lim and Thalmann used a simple curve simplification algorithm to extract key-frames in which error tolerance should be specified as the distance criterion. Error (threshold) parameters specify error requirement in the methods above and user could set these parameters according to different error requirements so as to acquire different data compression ratios. However, because of the difference in motion velocity and motion styles of different motion sequences, there is no exact one-to-one correspondence between error requirement and compression ratio. Consequently, for a user paying attention to key-frame number (compression ratio), different error parameters should be experimented repeatedly to meet the specified error requirement in the key-frame extraction. This is time-consuming.

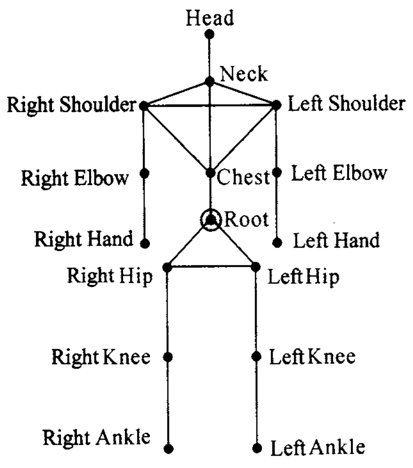
Key-frame set of motion data, as special temporal signal, can successfully abstract the original data sequences. Moreover, comparability is required between key-frame sets of similar motion types for the convenience of subsequent work such as indexing and editing. However, all the key-frame extraction methods mentioned above have shortcomings in motion feature expression in that they fail to take into account the structure information of the original motion sequences, thus lose the physical characteristics of motion data. So, key posture clusters respectively extracted from two motion sequences of similar motion types cannot meet the requirement of comparability. Furthermore, further processing such as indexing and editing will be influenced by the unsatisfied key-frame collections resulting from these methods.

Based on the analysis above, a novel layered curve simplification algorithm is developed for key-frame extraction. In this algorithm, motion features are described by bone angle which is defined as the angle between the limb bone and the central bone. Then according to the motion trajectory of bone angles, some possible extreme postures are chosen as key-frame candidates. Candidate key-frames should be refined by the layered curve simplification algorithm according to different error requirement before gaining final key-frame collection. A self-adaptive error parameter extraction method is also developed for the users who care only about compression ratio.

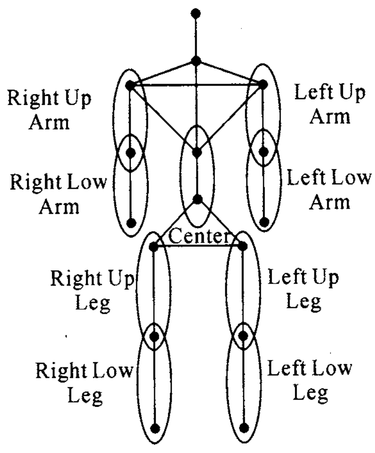
6.2.2 Key-frame Extraction from MoCap Data Based on Layered Curve Simplification Algorithm

6.2.2.1 Motion Describing Model

A simplified human skeleton model is defined as Fig. 6.12(a), which con-



(a) human skeleton



(b) skeleton segments

Fig. 6. 12 Human skeleton and segments

tains 16 joints that are constructed in the form of tree. Joint *root* is root of the tree and those paths from root to all endmost joints in human skeletal model form sub-trees of root. Joint root is represented by a 3D translation vector and a 3D rotation vector. The translation of joint root determines the current position of the skeleton while the rotation of joint root determines the overall orientation of the skeleton. For the other joints, rotation describes their orientation in the local coordinates of their parent joints. The factors all above codetermine the posture of human body.

Motion data M can be viewed as human posture sequence sampled temporally and discretely, where every sampling point as a frame described by 16 joints. In this way, at arbitrary time i, human body posture can be $\boldsymbol{F}_i=[\boldsymbol{p}_i^{(1)},\boldsymbol{r}_i^{(1)},\boldsymbol{r}_i^{(2)},\cdots,\boldsymbol{r}_i^{(16)}]$, in which, $\boldsymbol{p}_i^{(1)}\in\mathbf{R}^3$ and $\boldsymbol{r}_i^{(1)}\in\mathbf{R}^3$ describe the position and orientation (i. e. translation and rotation) of the root respectively. $\boldsymbol{r}_i^{(j)}\in\mathbf{R}^3, j=2,\cdots,16$ describes the orientation (rotation) of non-root joints. At arbitrary time i, the position of non-root joint N_j can be attained by 3D translation in form of the following equation:

$$\boldsymbol{p}_i^{(j)}=\boldsymbol{T}_i^{(\text{root})}\boldsymbol{R}_i^{(\text{root})}\cdots\boldsymbol{T}_0^{(\text{grandparent})}\boldsymbol{R}_i^{(\text{grandparent})}\boldsymbol{T}_0^{(\text{parent})}\boldsymbol{R}_i^{(\text{parent})}\boldsymbol{p}_0^{(j)} \tag{6-3}$$

where $\boldsymbol{p}_i^{(j)}$ is the world coordinate value of joint N_j at time i. $\boldsymbol{T}_i^{(\text{root})}$, $\boldsymbol{R}_i^{(\text{root})}$ are the translation matrix and rotation matrix of root joint at time i and they are given by $\boldsymbol{p}_i^{(1)}$, $\boldsymbol{r}_i^{(1)}$ respectively; $\boldsymbol{T}_0^{(k)}$ is the translation matrix of joint N_k (N_k is an arbitrary joint from root joint to N_j joint in the skeleton model) generated by the offset in the local coordinate of its parent joint. $\boldsymbol{R}_i^{(k)}$ is the rotation matrix of joint N_k (N_k is same as above) at time i and is generated by $\boldsymbol{r}_i^k$. $\boldsymbol{p}_0^{(j)}$ is the offset of N_j in the coordinate of its parent joint at initial time.

6.2.2.2 Motion Feature Representation

Generally speaking, motion features are extracted from information related to joints, including coordinate position, angular velocity, relative angle of joints as well as the orientation of son joints articulated to a certain joint.

According to this human skeleton, each frame of motion sequence is represented as a vector with 51 dimensions. Key-frame extraction directly from the original data space will give rise to low efficiency. Moreover, the intrinsic part of motion sequence is hidden by the high-dimensionality and will cause inconvenience in the follow-up motion analysis. Consequently, we need a key-frame extraction method that can keep the intrinsic feature of motion data.

Based on the human skeleton, we upgrade our motion describing method from joint-layer to skeleton layer. Considering that limbs move more frequently than other parts of human body, eight bones in limbs are extracted to represent motion feature. Therefore, nine bones are extracted as the objects to represent motion feature, including eight bones in human limbs and a central bone that is connected by *root* and *chest* joints as a reference bone (see Fig. 6. 12 (b)). Each bone is defined as a vector from the upper joint to the lower joint in human skeleton. Given a bone $\boldsymbol{B}^{(k)}$ $(1 \leqslant k \leqslant 9)$ connected by point N_i and N_j, it is defined as: $\boldsymbol{B}^{(k)} = \overrightarrow{N_iN_j} = \boldsymbol{p}^{(j)} - \boldsymbol{p}^{(i)}$, where N_i is the parent of N_j in human skeleton, and $\boldsymbol{p}^{(i)}$ and $\boldsymbol{p}^{(j)}$ are coordinates of N_i and N_j in the world coordinate system respectively.

For every limb bone, bone angle is defined as the angle between the limb bone and the central bone. Given a limb bone, its bone angle at the ith frame is defined as follows:

$$\theta_i^{(k)} = \cos^{-1}\left(\frac{\boldsymbol{B}_i^{(k)} \cdot \boldsymbol{B}_i^{(\text{center})}}{|\boldsymbol{B}_i^{(k)}| \cdot |\boldsymbol{B}_i^{(\text{center})}|}\right) \quad (k=1,\cdots,8) \tag{6-4}$$

where $\boldsymbol{B}_i^{(\text{center})}$ represents the central bone at the ith frame and θ is in the interval $[0, \pi]$.

Consequently, by calculating 8 bone angles, the ith frame of human motion is represented by a eight-dimensional angle vector: $\boldsymbol{F}_i = (\theta_i^{(1)}, \cdots, \theta_i^{(8)})$. This motion feature representation not only reduces the dimensionality of original motion data, but also efficiently represents the physical feature of human motion.

6.2.2.3 Key-frame Extraction

With all the work above, now we can represent human motion data by 8 bone angle vectors. Consequently, every frame of motion data corresponds to a point in the 8D bone angle space, and motion trajectory can be viewed as a curve in the 8D bone angle space which is formed by connecting these points in the sequence of time. In this way, to get key-frame collection from a certain motion sequence is to simply find a set of points on the curve. Polygonal curve formed by these chosen points can then be used to approximate the original motion curve. Then curve simplification is employed to get the key-frame set. However, if we take curve simplification directly, although efficiency will be improved in that the feature representing method proposed in this book has advantages in its low dimensionality, some "extreme" postures will be missed to some extent. So we make possible extreme positions as key-frame candidates, and finally get key-frame collection that not only abstract motion data but also retain motion features.

Candidate Key-frame Searching

It is found that extreme values exist in the trajectories of bone angles. Fig. 6. 13 illustrates the trajectories of bone angles of right leg that is composed of right upper leg and right lower leg in a walk motion with 20 frames. There are 3 local extreme points at the 8th, 12th and 17th frames respectively, which correspond to the extreme postures that occur in this motion sequence. At the 8th frame, the right leg reaches the highest position, at the 12th frame it reaches the ground for the first time and at the 17th frame the gravity center of human is upon right leg. We can see that the changes of bone angles of right leg describe the movement of right leg in motion sequence well. The postures at those extreme points can be selected as candidate key-frames because they are the most informative representatives of right leg's movement.

Similarly, changes of other limbs' bone angles can well describe the movement of corresponding limbs. According to all bone angles' changes, candidate key-frames can be obtained through collecting those frames at which local extreme points occur. Since those candidate key-frames are the union of extreme postures in the movements of all human limbs, they comprise most of all possible extreme postures in motion sequence.

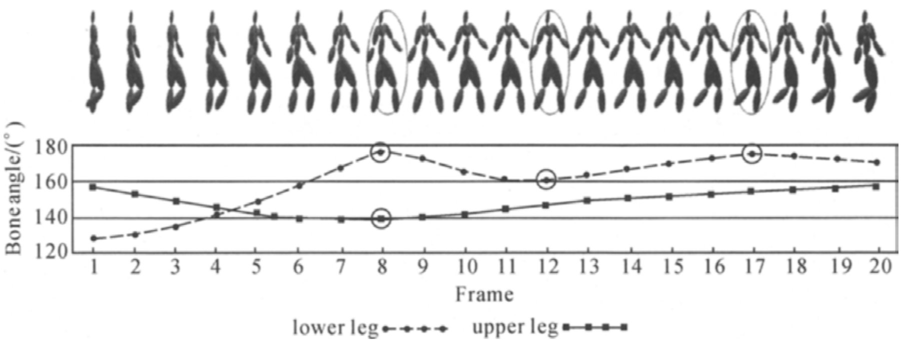


Fig. 6. 13 Determination of candidate key-frame

Key-frame Refinement Based on Layered Curve Simplification

In the process of human's movement, there are two phenomena:

- The extreme points of different bone angles may not occur at the same time, but in a certain time interval;
- There is some noise when human motions are being captured, which leads to some posture's distortions.

These phenomena result in that there are some candidates near to each other. For lower error requirement or higher compression ratio, those candidates should be merged. On the other hand, due to the existence of some motion clip with less bone angles' change, there are some adjacent candidates that are distant in the time sequence. For higher error requirement or stronger ability to summarize human motion, one or more frames between those candidates should be selected as key-frames. Consequently, candidate key-frames should be refined according to different error requirements before gaining the final key-frame collection.

Before elaborating on our refinement algorithm, let's take a look at Simple Curve Simplification (SCS) algorithm [18,22]. The main idea of the algorithm is to choose a subset from original data set which lies in a high-dimensional space formed by connecting all the data points. Then the polygonal curve is approximated using this subset. We judge whether the approximating is finished by calculating maximum value of straight-line distances from all data points on the curve to the line segment connecting two ends of the curve. Then we judge whether the ratio of this maximum value and length of the line segment meet the error requirement. Fig. 6. 14 gives a demonstration of the algorithm. The algorithm can be generalized to higher dimensionality for the ratio of maximum distance to length of line segment is taken as approximation judging condition.

Inspired by the SCS algorithm, we propose a Layered Curve Simplification (LCS) method to refine our candidate key-frames. The LCS algorithm is described as follows.

Given a sequence data set M: $M=\{F_i \mid i=1,\cdots,N\}$, $F_i=(x_i^{(1)},\cdots,x_i^{(m)})$, sequentially connect data points in M to get a curve in m-dimensional space; Given a set C: $C\subset M$, $F_1, F_N\in C$, try to find a sequence data set K which approximates the curve of M under certain error requirement, where $K\subset M$ and the elements in K should be in C as many as possible. To solve this problem, M and C can be treated as curves in m-dimensional space and they can be combined and constructed into a two-layer structure. The higher layer C is, the lower layer M will be. In the beginning, we run SCS on the upper layer to find K. If the resulting approximation cannot satisfy the error requirement, then SCS runs on the lower layer. Those newly gained points at which curve is sub-divided would be inserted into the upper layer. Then, SCS runs to the upper layer again. This procedure

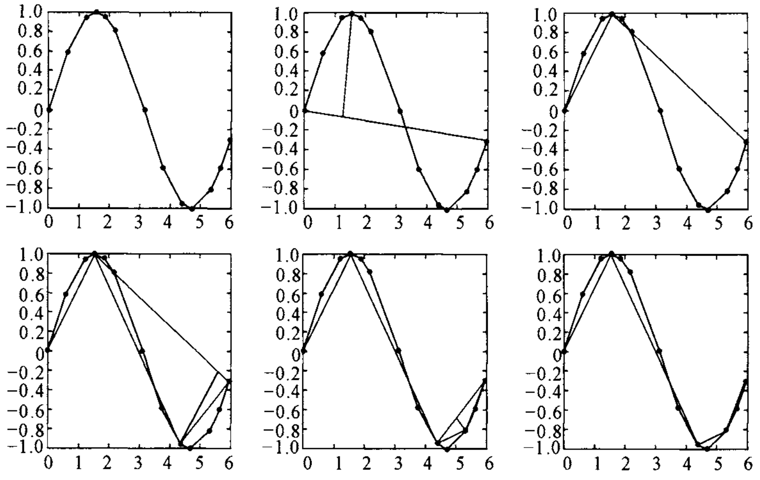


Fig. 6.14 Simple curve simplification (SCS) algorithm

is recursively repeated until the resulting approximation satisfies the error requirement specified for the given distance criterion. By running the SCS method to each layer alternately, the data set K would be gained finally. Two arrays (*c_num* and *pair*) are used in this algorithm: "*c_num*" sorted by value ascendingly records the position in M of every element in C, while "*pair*" marks whether the elements in M belong to set K. If a certain element in M does not belong to K, set its corresponding element in "*pair*" to value 0, otherwise, its corresponding element in "*pair*" contains a point pointing to the position in M of the next element in K. In SCS, the shortest straight-line distance from the ith element to the line connected by n_1 and n_2 can be got from the formula below:

$$d_i = \left[\sum_{k=1}^{m} (x_i^{(k)} - x_{n_1}^{(k)})^2 - \frac{\left(\sum_{k=1}^{m} (x_{n_2}^{(k)} - x_{n_1}^{(k)})(x_i^{(k)} - x_{n_1}^{(k)}) \right)^2}{\sum_{k=1}^{m} (x_{n_2}^{(k)} - x_{n_1}^{(k)})^2} \right]^{1/2} \tag{6-5}$$

6.2.2.4 Layered Curve Simplification Algorithm

Input: Sequence data set M: $M=\{F_i \mid i=1,\cdots,N\}$, $F_i=(x_i^{(1)},\cdots,x_i^{(m)})$; set C: $C\subset N$ and $F_1, F_N\in C$; error parameter $\delta\in \mathbf{R}^+$

Output: Array *kfSet*: $kfSet=\{i \mid F_i\in M \& F_i\in K,\ i=1,\cdots,N\}$

Step 1 Initialization. N is the number of elements in set C; $c\_num=\{i \mid F_i\in M\ \&\ F_i\in C,\ 1\leqslant i\leqslant N\}$. $pair(1)=N$, $pair(N)=1$, *trackMark*$=1$, this variable is to mark the position of current element in processing in set C. *lastMark*$=0$, this variable is to mark the position of the element in set C that has been processed most recently.

Step 2 If $pair(trackMark)=1$, go to Step 5, otherwise calculate the subset of set C which contains all elements between $trackMark$ and $pair(trackMark)$: $subset=\{c\_num(i) \mid trackMark<c\_num(i)<pair(trackMark),\ 1\leqslant i\leqslant N\}$, if size of $subset>0$, go to Step 3, otherwise go to Step 4.

Step 3 SCS on the upper layer

(1) Calcul ate the distances from every point in subset to line $|F_{pair(trackMark)}F_{trackMark}|$ according to equation(4-1). max_dist stores the maximum distance, and its corresponding element is max_ind.

(2) Calculate $ratio=max\_dist/\|F_{pair(trackMark)}F_{trackMark}\|$. If $ratio>\delta$, let $pair(max\_ind)=pair(trackMark)$, $pair(trackMark)=max\_ind$. Otherwise, $trackMark=pair(trackMark)$.

(3) Go to Step 2.

Step 4 SCS on the lower layer

(1) Let $subset=[trackMark : pair(trackMark)]$.

(2) Calculate the distances from every point in subset to line $|F_{pair(trackMark)}|$ according to equation (4-1). max_dist stores the maximum distance, and its corresponding element is max_ind.

(3) Calculate $ratio=max\_dist/\|F_{pair(trackMark)}F_{trackMark}\|$. If $ratio>\delta$, $C=C\cup\{F_{max\_ind}\}$, reset c_num and n, let $pair(max\_ind)=pair(trackMark)$, $pair(trackMark)=max\_ind$, go to Step 2. Otherwise, $trackMark=pair(trackMark)$.

(4) Go to Step 2.

Step 5 $kfSet=\{i \mid pair(i)\neq 0,\ 1\leqslant i\leqslant N\}$

To use the layered curve simplification algorithm to refine candidate key-frames, let the motion data represented by 8D bone angle vectors be set M and let candidate key-frame collection be set C, and the resulting set K is the final key-frame collection that satisfies users' error requirement. In this method, curve simplification in the higher layer is equal to merging adjacent candidate key-frames, and operation in the lower layer is equal to selecting new key-frame. Since key-frames are mostly extracted from the candidate key-frame collection that is the higher layer in the two-layer structure, it is confirmed that the resulting key-frames contain those extreme posture as many as possible. Thus we can get good abstract of the original motion data while maintaining motion characteristics, and comparison between similar motions also becomes possible.

6.2.2.5 Adaptive Extraction Parameters

Using the layered curve simplification based key-frame extraction proposed in the above section, key-frame collections of motion sequences satisfying different error requirements can be obtained through specifying different parameters of error requirement. But in most applications users only have idea about the number of key-frames and do not care about the value of extraction parameter which results in the actual number of key-frames. Here we employ adaptive extraction parameters to solve this problem (see Fig. 6.15).

The main idea of adaptive extraction parameters is to adjust the parameters of error requirement δ automatically according to the difference between the actual value and the desired value of the number of key-frames. If the actual value is more than the desired value, δ increases and the increase rate is δ_inc ($\delta\_inc \in (0,1)$). Otherwise, δ decreases and the decrease rate is δ_dec ($\delta\_dec \in (0,1)$). There are several factors that affect the convergence of the number of key-frames as follows:

- Greater initial value of δ and smaller change rate of δ lead to slow convergence of the desired key-frame number;
- Greater change rate of δ lead to the oscillation of actual value around the desired value of the number of key-frames severely;
- The actual value is oscillating around the desired value of the number of key-frames in a small neighborhood, but the actual value cannot converge completely.

To solve the first two problems above, we adjust the change rate of δ, $d\delta$ ($d\delta \in \{\delta\_inc, \delta\_des\}$, $d\delta \in (0,1)$) automatically. If $d\delta$ is in the same direction in two consecutive loops, $d\delta$ increases to speed the change of $d\delta$.

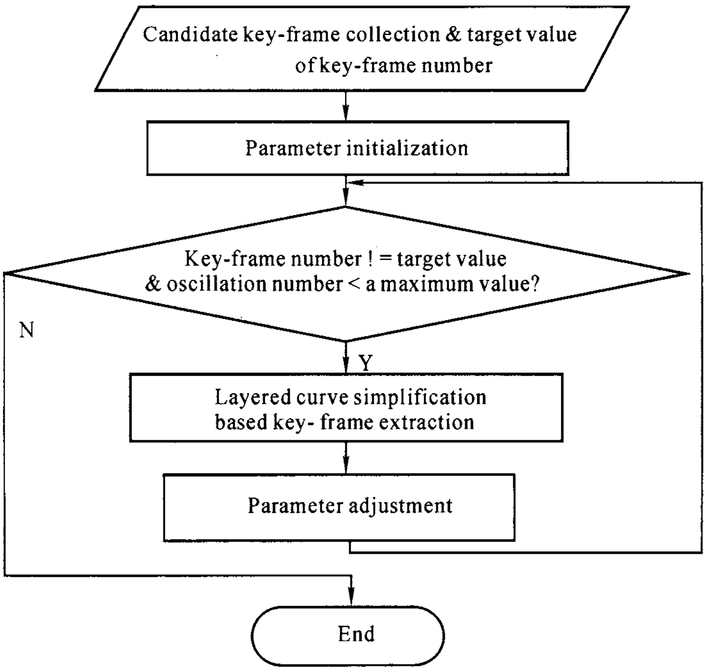

Candidate key-frame collection & target value of key-frame number
Parameter initialization
Key-frame number ! = target value & oscillation number < a maximum value?
N
Y
Layered curve simplification based key- frame extraction
Parameter adjustment
End

Fig. 6.15 Flowchart of self-adaptive parameter extraction

And when the actual value is oscillating around the desired value, $d\delta$ decreases to relieve the oscillation. The new $d\delta$ can be calculated as follows:

$$f_{\text{inc}}(d\delta)=\sqrt{1-(d\delta-1)^2}, \quad f_{\text{dec}}(d\delta)=1-\sqrt{1-d\delta^2}$$

In addition, a maximum oscillation number will be defined as a condition to jump out from the loop in order to solve the above third problem.

6.2.3 Results and Discussions

We captured more than 100 real human motion sequences with different motion types at 60 Hz frame rate as the testing collection to show the effectiveness of LCS. And it is implemented by Matlab and run on an Intel PentiumIV 2.6 GHz CPU with 512 MB memory.

6.2.3.1 Key-frame Extraction

Firstly, key-frames are extracted from all kinds of motion sequences. As shown in Fig. 6.16, extreme postures can be extracted from motion sequences with different motion types precisely. For example, in Fig. 6.16 (a) six key-frames are extracted from a kicking motion with 90 frames. The 1st key-frame is a ready pose, at the 40th frame the knee joint lifts to the highest point and the foot joint reaches to the furthest point at the 49th frame, and in the 90th frame subject moves back to the ready pose again. The 31st and 69th frames are the important transitional postures among these extreme postures.

Fig. 6.17 shows key-frame collections with different numbers of key-frames for step-up motion. If the desired key-frame number is large, we get a key-frame collection with some redundant frames (bottom row). When we set the desired key-frame number smaller and smaller, this method can get all of the extreme and important transitional postures precisely and the redundant frames are pruned (second row). But if we set the

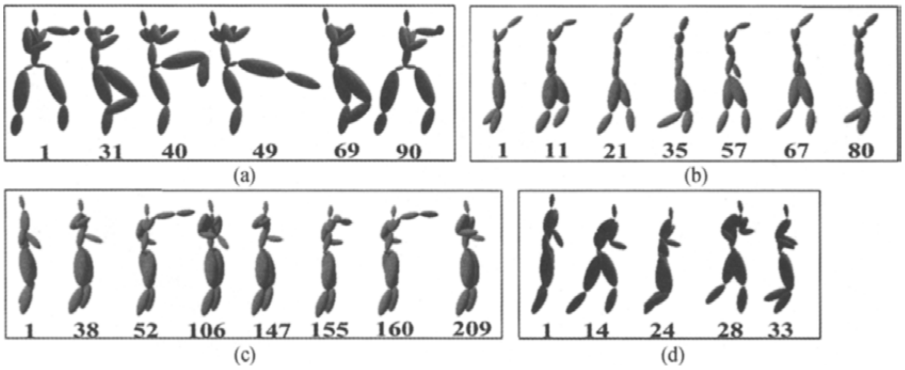


Fig. 6.16 Key-frame sequences extracted from four dissimilar human motions. (a) Kick; (b) Wave and walk; (c) Punch; (d) Run

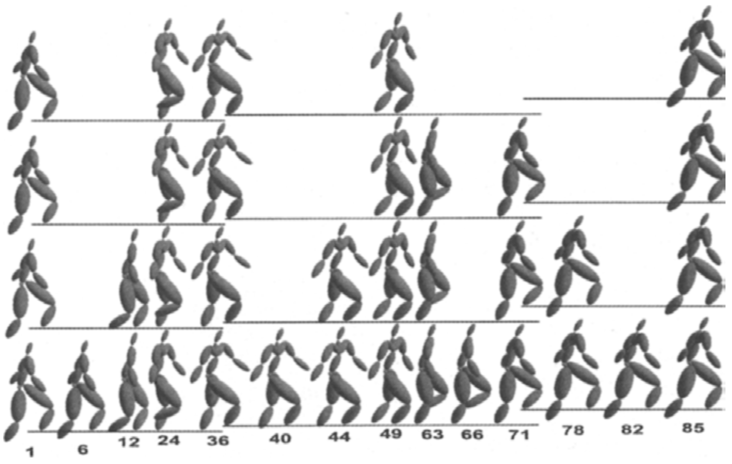


Fig. 6. 17 Key-frame collections for step-up motion with different numbers of key-frames

key-frame number smaller further, some important key-frames will be missed (in top row, the 71st frame is missed).

6.2.3.2 Comparison with Other Key-frame Extraction Methods

In Fig. 6. 18, one non-periodical motion, jump-up, is demonstrated to make a comparison among these three key-frame extraction methods. The jump-up motion sequence consists of 91 frames and we preset the desired number of key-frames to 12. Before the 30th frame, the subject moves little. He begins to jump at the 48th frame and then he jumps onto a higher ground. The results of three methods are shown in Fig. 6. 18. It can be seen that under the same compression ratio, LCS achieves the best result,

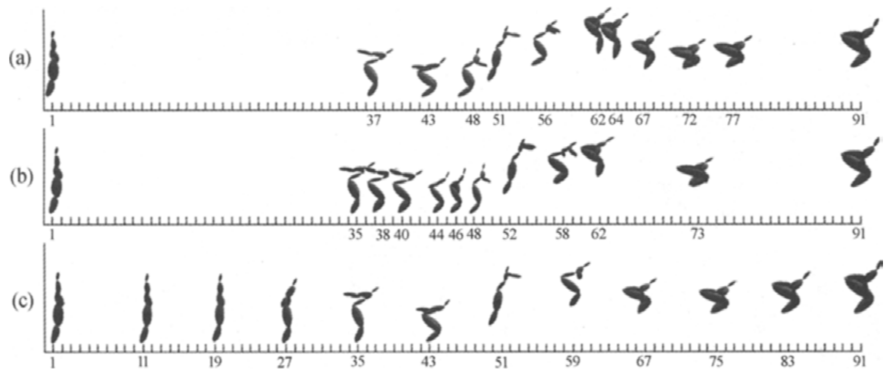


Fig. 6. 18 Key-frame sequences extracted from a non-periodical motion, jump-up, by using three different key-frame extraction methods under the same compression ratio. (a) Layered curve simplification based method; (b) Simple curve simplification based method; (c) Uniform sampling method

and the result of SCS is better than that of Uniform Sampling (US). US method loses many details of human motion and cannot describe the original motion effectively due to over-sampling under less posture changes and under-sampling under intense posture changes. SCS describes the original motion better than US, but its key-frame collection misses some extreme postures such as the 67th frame when the subject touches the ground for the first time compared to the key-frame collection extracted by LCS.

6.2.3.3 Key-frame Extraction from Similar Motions

In Fig. 6. 19 four similar motions are shown to demonstrate the consistency among similar motions. Each of them contains a complete cycle of walking movement which starts and ends at the same phase, but they have different motion styles and velocities. The lengths of these four motions are 38, 40, 39 and 68 respectively and the desired number of key-frames is set to 7 for all of them. In Fig. 6. 19 (a), the subject is swinging his arm excessively as he walks, while the subject's arms move little in Fig. 6. 19 (b). In Fig. 6. 19 (c), the subject's body is leaning backwards as he walks, and in Fig. 6. 16 (d), the subject is shambling. In all of key-frame collections the Center of Gravity (COG) is on the left foot at the 1st key-frame, the right foot touches the floor for the first time at the 3rd key-frame, the COG transfers to left foot at the 4th key-frame, at the 6th key-frame, the right foot touches the floor again and the COG transfers back to right foot again at the last key-frame. The 2nd and 5th key-frames contain the important transitional postures among these extreme postures. It can be seen that these four key-frame sequences correspond to each other basically, so it is possible for further processing such as motion retrieval, matching and motion editing. In conclusion, experimental results show that when LCS based key-frame extraction method is applied on similar motion sequences

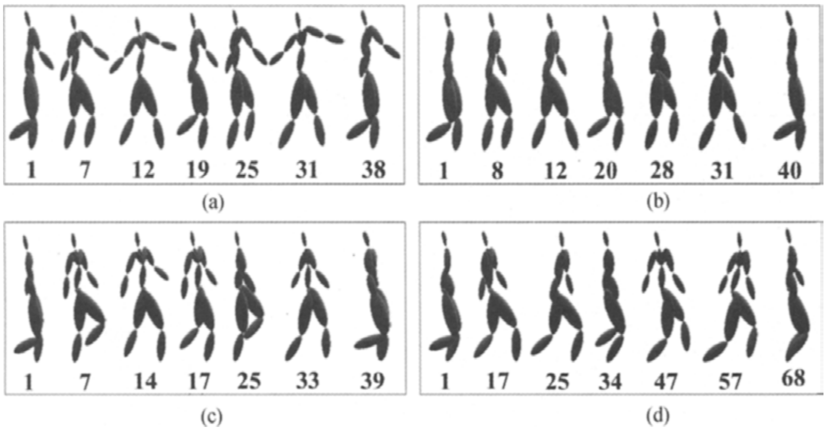


Fig. 6. 19 Key-frame sequences extracted from four similar human motions

with different lengths, consistency can be retained among the key-frame collections.

6.2.3.4 Adaptive Extraction Parameters

Table 6.2 shows the initial parameters and results of extraction in the experiments. M1, M2 and M3 correspond to the (d), (b), (c) motions in Fig. 6.16, and the maximum of the oscillation number is 100. The changes of δ, δ_*inc* and δ_*dec* in Experiment 1 and Experiment 5 are shown in Figs. 6.20 (a) and (c), respectively. Fig. 6.20 (b) is the zoom in of a part of Fig. 6.20(a) where epoch (loop number) is from 6 to 28. In Fig. 20(a), initial δ is greater (100) and initial δ_*inc* and δ_*dec* are smaller (0.01). The algorithm increases δ_*dec* to speed δ's decrease at the 6th loop, where δ decreases too much and needs increasing. After that, the algorithm makes δ_*inc* increase to speed δ's increase. At the 14th loop, δ begins to decrease. Such process causes δ to oscillate around its target value. Since every parameter is adjusted automatically, the amplitude of δ's oscillation will become smaller and smaller and at the 28th loop, δ is equal to its target value and algorithm terminates. While in Experiment 5 (Fig. 6.20 (c)), the value of key-frame number cannot be convergent completely, which causes δ always to oscillate around its target value. So we terminate the algorithm by setting the maximum oscillation number, and the value that is most close to target value is our final result.

Table 6.2 shows that the convergence of the number of key-frames is independent of initial parameters. Even for those that have too many frames and reach the maximum of oscillation number (Experiment 5), our algorithm's running time is acceptable. Consequently, initial values of the algorithm's parameters can be defined at random in their range of value. It is noticed that δ is often in the interval (0,1) when the algorithm terminates, so the initial value of δ is usually defined randomly in the interval (0,1). Defining initial parameters at random automatically relieves user's work.

Table 6.2 Initial parameters and results of extraction

ID	Motion (frames)	# Key-frames (desired/ actual)	δ (initial/ actual)	δ_*inc* (initial/ actual)	δ_*dec* (initial/ actual)	# Loops	Time	# Oscill-ation
1	M1(33)	5/5	100/ 0.4970	0.01/1 *	0.01/ 0.1411	28	0.1720	8
2	M1(33)	5/5	0.01/ 0.4852	0.90/1 *	0.90/ 0.5641	11	0.0620	2
3	M2(80)	7/7	1.0/ 0.4149	0.90/0.9	0.01/ 0.8729	4	0.0150	0
4	M3(209)	8/8	1.0/ 0.2738	0.01/ 0.9919	0.90/ 0.5641	16	0.3590	6
5	M3(209)	20/19	0.5/ 0.1650	0.01/ 0.1411	0.90/ 0.1743	197	3.5470	100

* Actual value of δ_*inc* approaches 1 but is not equal to 1.

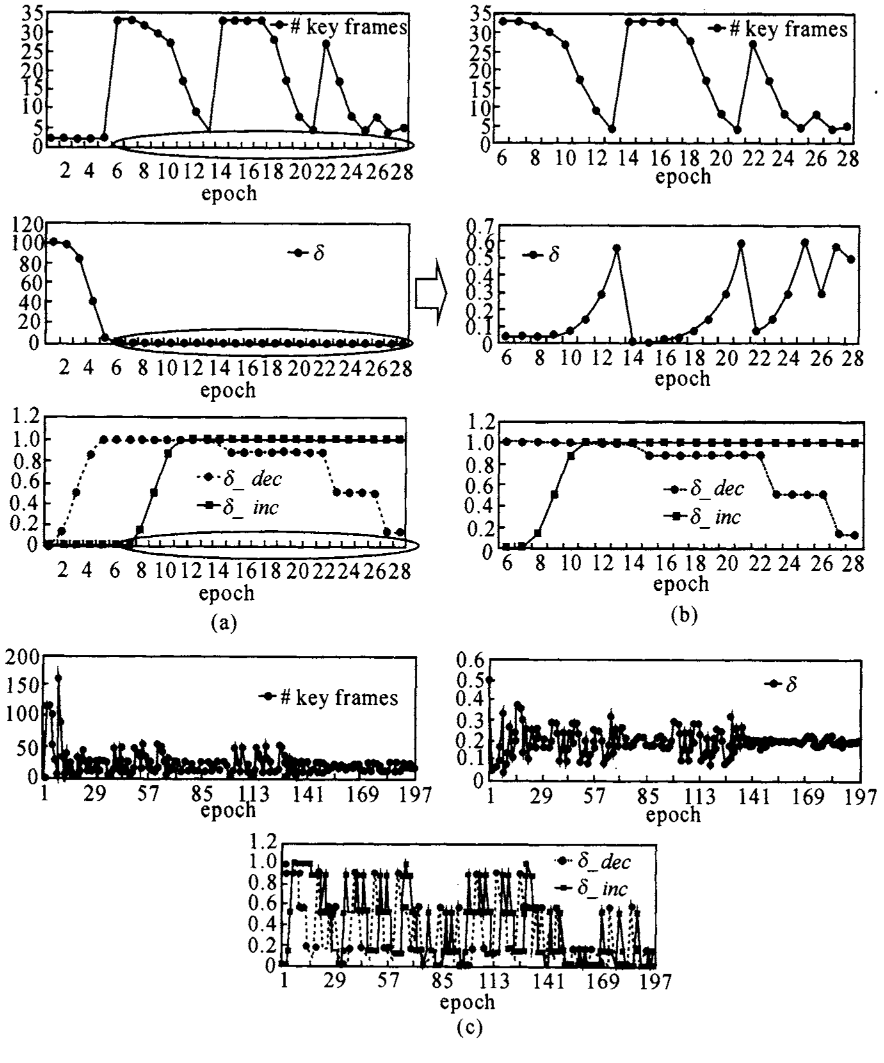


Fig. 6. 20 Algorithm parameters graph. For (a) and (c), upper-graph illustrates changes of the number of key-frames, middle-graph illustrates changes of δ and lower-graph illustrates changes of δ_ *inc* (solid line) and δ_ *dec* (dotted line); (b) is a zoom in of circled part in (a) where the epoch is over an interval [6, 20]

6.2.3.5 Motion Compression and Reconstruction

Fig. 6. 21 compares the original motion data with the reconstructed data which are generated from key-frames extracted by our method. Before the 40th frame, punch motion is intense so more key-frames are extracted, which is very important to preserve the details of the original motion in reconstruction. After the 40th frame, punch motion becomes gentle so

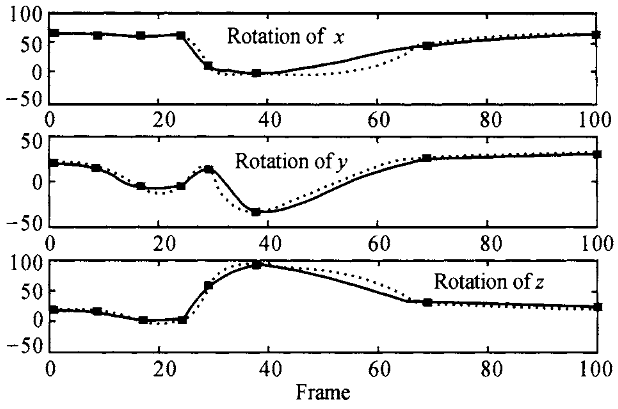


Fig. 6. 21 Motion reconstruction. The dot lines represent the original x, y, and z rotation values of right shoulder joint in a punch motion. The block is the key-frame and solid line is the reconstructed motion data. Here we use the piecewise cubic Hermite interpolation polynomial algorithm for reconstruction[23]

sparse key-frames are extracted. Fig. 6. 22 shows the reconstructed sequence of this punch motion.

Additionally, we have experimented on long motion sequences in order to get some relation between compression ratio and reconstruction. In Fig. 6. 23, we can see that more extracted key-frames (lower compression ratio) result in lower reconstruction error. From all of the experimental results on the testing collection (more than 100 motion sequences), compression ratio with value of 10 is the best tradeoff for quality of motion reconstruction.

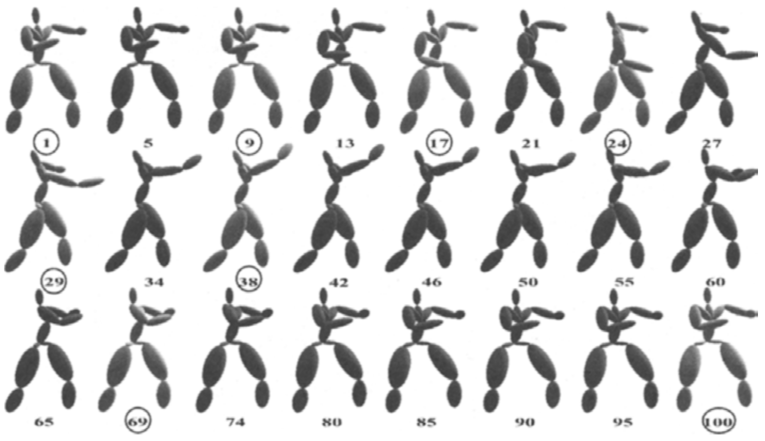


Fig. 6. 22 Reconstructed sequence of punch motion. Numbers with circle represent the key-frames. Others represent the postures which are reconstructed from key-frames

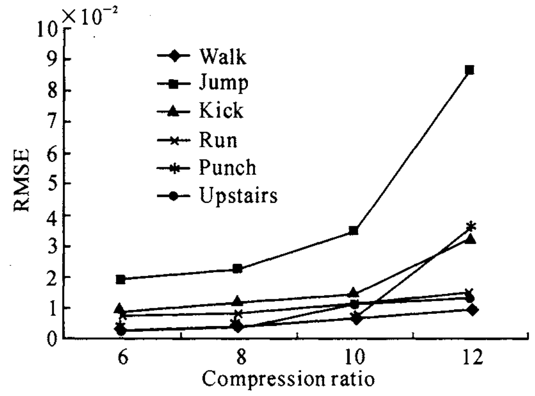


Fig. 6.23 Reconstructed error at different compression ratios

In this section, we have presented a method for efficient key-frame extraction for human motion capture data. One main contribution of this work is the introduction of bone angles as human motion feature representation, by which human motion's extreme postures are searched for regardless of the motion type and style. The second contribution is the proposal of layered curve simplification method to refine candidate key-frame collection that is composed of extreme postures. Since the final key-frame collections are selected based on extreme postures, they have corresponding key-frames for those similar motions, which benefits comparison among the similar motions and can be used for key-frame based motion retrieval application directly. Thirdly, adaptive extraction parameters method is proposed. By this method, given a specified compression ratio, key-frames can be extracted without specifying any extraction parameter.

This key-frame extraction method can be used in multiple applications including: (1) motion summarization for browsing; (2) key-frame based motion retrieval; (3) motion compression and reconstruction; (4) key-frame based motion synthesis.

Experimental results show that key-frame collections attained from layered curve simplification algorithm can avoid the problems of over-sampling and under-sampling which often occur in uniform sampling method. Moreover, since layered curve simplification method is applied to refine extreme posture key-frames, limitations in simple curve simplification algorithm in key-frame extraction could be avoided, which means some important extreme postures will not be missed and consistency will be remained among key-frame collections of similar motion sequences. Furthermore, with adaptive extraction parameters, users only need to pay attention to key-frame extraction results without worrying about the parameter values.

6.3 Motion Data Retrieval

With the development of motion capture techniques, more and more 3D motion libraries become available. When these 3D motion libraries are put into use, a content-based 3D motion retrieval technique is an urgent need. The reasons are mainly related to the following two aspects.

- It is an important tasks to retrieve required motions from existing 3D motion libraries for the purpose of developing more efficient animation tools. Recently, in the field of motion capture based animation, researchers have focused on the development of semantic processing tools for motion data in abstract layer. For example, in SIGGRAPH 2002, Pullen and Bregler [24] presented a motion capture assisted animation, which automatically retrieves appropriate motion fragments from an existing motion library according to a given key-frame motion; in SIGGRAPH 2003, Dontcheva proposed a motion directed based animation algorithm to retrieve similar motion clips from motion libraries according to animator's motion sample.
- Content-based 3D motion retrieval technique also facilitates animators to efficiently manage motion libraries. Traditional text-base retrieval techniques can hardly be employed for motion retrieval because it is time-consuming and inaccurate for users to annotate manually.

Content-based retrieval technique has been hot in multimedia and signal processing field and is widely used in multimedia technology such as image, audio and video. In this technique, visual and audio characters (such as color, texture, motion characters) are extracted to describe the content of video (audio). Then retrieval can be realized between similar video streams or audio streams according to the sample submitted by users.

Content-based retrieval techniques have many advantages over traditional retrieval techniques; however, there is still no efficient content-based retrieval technique for 3D motion at present.

The following challenges have to be addressed for 3D motion retrieval techniques: (1) Effective reduction is required for high-dimensional motion data; (2) Motion signal is the combination of signals from every joint, which demands appropriate motion feature descriptors; (3) It is time-consuming to calculate the similarity or distance between two motions.

Aiming at the above challenges, a content-based 3D motion retrieval algorithm, 3D motion retrieval with motion index tree, is presented as below.

In 3D motion retrieval with motion index tree, a hierarchical motion description is adopted to represent a posture, based on which motion library can be partitioned hierarchically and construct a motion index tree to facilitate motion retrieval. Nearest Neighbor (NN) rule-based dynamic clustering algorithm is employed to partition the motion library and construct the motion index tree. The similarity between two motions is calculated by elastic match. To improve the efficiency during the similarity calculation, we adopt an adaptive clustering-based key-frame extraction algorithm to extract posture key-frame sequences, which are used in elastic match. During the retrieving stage, the motion index tree serves as a hierarchical classifier to determine the sublibrary that contains the promising similar motions to the query sample. Next, key-frame posture sequences are extracted from the sample and the motion from the sub-library respectively, and the similarity between them is calculated using elastic match.

6.3.1 Motion Index Tree

Within the hierarchical motion description, the motion of a parent joint may induce child joints, whereas the motion of a child joint is unable to influence its parent joint. Obviously, the joints at high levels of the hierarchy are more significant than those at lower levels in terms of determining a motion. This hierarchy among the parameters of a posture can be used to facilitate motion retrieval by building a corresponding motion index tree for a motion library. Given the above human body as an example, all the joints can be divided into the following five levels according to their positions in the tree illustrated in Fig. 6.24, as {Root}, {L_Hip, R_Hip, Chest}, {L_Knee, R_Knee, Neck, L_Shoulder, R_Shoulder}, {L_Ankle, R_Ankle, Head, L_Elbow, R_Elbow}, and {L_Wrist, R_Wrist} from top to bottom. We construct a motion index tree based on this hierarchy to partition and structure the 3D motion library, as shown in Fig. 6.24.

The main steps to construct the motion index tree are outlined as follows.

Step 1 Partition the 3D motion library ML into several subsets $ML_{2;k}$, using dynamic clustering according to the motion of the first-level joint (viz. {Root}), and build a sample motion set from the subsets for the node Root. Particularly, motions closest to the cancroids of each subset are selected as samples from this subset.

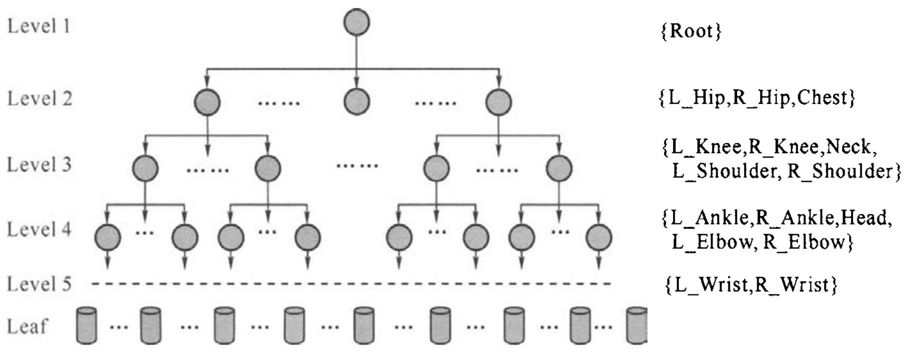

Level 1
Level 2
Level 3
Level 4
Level 5
Leaf
{Root}
{L_Hip,R_Hip,Chest}
{L_Knee,R_Knee,Neck, L_Shoulder, R_Shoulder}
{L_Ankle,R_Ankle,Head, L_Elbow, R_Elbow}
{L_Wrist,R_Wrist}

Fig. 6.24 Motion index tree

Create a null node with an empty sample set, associate it with a subset $ML_{2,k}$, and take it as one of the child nodes of Root.

Step 2 Partition each subset $ML_{i,k}$ respectively according to the motion of joints at the level i in the hierarchy and build a sample set for its corresponding node in the motion index tree in the similar way in Step 1. Again, create a null node for a newly created subset $ML_{i+1,j}$ and take it as a child node of $ML_{i,k}$.

Step 3 Repeat Step 2 until all joints at the lowest level of the hierarchy are processed.

Step 4 Take the last partitioned subsets as leaf nodes of the motion index tree.

6.3.1.1 Nearest Neighbor Rule-based Dynamic Clustering

Partitioning a motion library into subsets is a well-defined clustering problem. As mentioned above, motion is a high-dimensional signal and no function is available to describe the probability density up to now. Due to the lack of available probability density functions, we partition the motion library based on the similarity between sample motions. Therefore, nearest neighbor rule-based dynamic clustering [25] is adopted to partition and structure the library.

Given two motion samples y_i and y_j, if y_j is the Ith Nearest Neighbor (NN) of y_i, the "NN coefficient" of y_j to y_i is defined as I. Likewise, if y_i is the Kth nearest Neighbor of y_j, the NN coefficient of y_i to y_j is defined as K. Let α_{ij} be the "NN value" between y_i and y_j, $\alpha_{ij}=(I+K-2)$. If M_i and M_j are classified into the same cluster, the connection cost is defined as α_{ij}; otherwise, the cost is 0. To eliminate clusters with only one sample, the connection cost of the self-connection is defined as $2N$ (N is the number of motions in the library). The objective function J_{NN} is defined as the sum of the total inner-cost L_{IA} and total inter-cost L_{IR}:

$$J_{\text{NN}}=L_{\text{IA}}+L_{\text{IR}}$$

The total inner-cost L_{IA} can be defined as the sum of all the connection values between every pair of samples in the whole library, given that the connection cost between samples from different clusters is 0.

$$L_{\text{IA}}=\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_{ij} \tag{6-6}$$

To calculate the inter-cost, the minimal NN value between cluster ω_i and ω_j, γ_{ij}, is computed first. And γ_i, the minimal NN value between cluster ω_i and all the other clusters can be calculated as follows:

$$\gamma_i=\min_{j\neq i}\gamma_{ij} \tag{6-7}$$

Let $\alpha_{i\max}$ and $\alpha_{k\max}$ be the maximal connection cost between samples in the cluster ω_i and ω_k respectively, c be the number of clusters, the inter-cost of cluster x_i to the other clusters is defined as follows:

$$\beta_i = \begin{cases} -[(\gamma_i - \alpha_{i\max}) + (\gamma_i - \alpha_{k\max})] & (\gamma_i > \alpha_{i\max},\ \gamma_i > \alpha_{k\max}) \\ \gamma_i + \alpha_{i\max} & (\gamma_i \leqslant \alpha_{i\max},\ \gamma_i > \alpha_{k\max}) \\ \gamma_i + \alpha_{k\max} & (\gamma_i > \alpha_{i\max},\ \gamma_i \leqslant \alpha_{k\max}) \\ \gamma_i + \alpha_{k\max} + \alpha_{i\max} & (\gamma_i \leqslant \alpha_{i\max},\ \gamma_i \leqslant \alpha_{k\max}) \end{cases} \tag{6-8}$$

The total inter-cost can be defined as the sum of all the inter-costs.

$$L_{\mathrm{IR}} = \sum_{i=1}^{c} \beta_i \tag{6-9}$$

The optimal clustering should result in the minimal objective value J_{NN}.

The following iterative method is proposed to solve this problem.

Step 1 Calculate the distance matrix $\boldsymbol{\Delta}$, with each element $\Delta_{ij} = \Delta(y_i, y_j)$, where Δ_{ij} is the distance between motion y_i and y_j.

Step 2 Construct NN coefficient matrix $\boldsymbol{M}$ based on $\boldsymbol{\Delta}$, with each element M_{ij} being the NN coefficient of y_j to y_i.

Step 3 Build NN value matrix $\boldsymbol{L}$ based on the NN coefficient matrix $\boldsymbol{M}$.

Step 4 Connect each element to the element, to which it has the minimal NN value in $\boldsymbol{L}$, and form the initial clusters.

$$L_{ij} = \begin{cases} M_{ij} + M_{ji} - 2 & (j \neq i) \\ 2N & (j = i) \end{cases}$$

Step 5 Calculate γ_i of each cluster and compare it with $\alpha_{i\max}$ and $\alpha_{k\max}$. If γ_i is smaller than either of $\alpha_{i\max}$ and $\alpha_{k\max}$, merge cluster ω_i and ω_k.

Step 6 Repeat Step 5 until no clusters can be merged.

6.3.1.2 Key-frame Extraction

Because each motion is represented as a frame sequence, the similarity between two motions is defined as that between the two corresponding frame sequences. However, even a short motion of 4 seconds is composed of approximately 100 frames. It is time-consuming to calculate the similarity with these original data. To improve the computational efficiency, we extract key-frame sequences from two motions and calculate the similarity between the key-frames only. Taking the motion of the second level joints as an example, we show the key-frame extraction algorithm in detail below.

Given a motion $\boldsymbol{M}$ with N frames, the motion at the second level can be represented as a $(3\times3\times N)\times1$ vector $\boldsymbol{M}_2$:

$$\boldsymbol{M}_2 = [\boldsymbol{F}_1, \boldsymbol{F}_2, \cdots, \boldsymbol{F}_N]$$

$$\boldsymbol{F}_i = [r_{\mathrm{l}xi}, r_{\mathrm{l}yi}, r_{\mathrm{l}zi}, r_{\mathrm{r}xi}, r_{\mathrm{r}yi}, r_{\mathrm{r}zi}, r_{\mathrm{c}xi}, r_{\mathrm{c}yi}, r_{\mathrm{c}zi}]$$

where $\boldsymbol{F}_i$ is the ith frame at the second level, $r_{\mathrm{l}xi}, r_{\mathrm{l}yi}$, and $r_{\mathrm{l}zi}$ are the rotation parameters of joint Left Hip; $r_{\mathrm{r}xi}, r_{\mathrm{r}yi}$, and $r_{\mathrm{r}zi}$ are the rotation parameters of joint Right Hip; r_{cxi}, r_{cyi}, and r_{czi} are the rotation parameters of joint chest.

At present, many methods have been proposed for key-frame extraction in the field of video abstraction. However, due to the periodicity of motions, most of these algorithms, including sampling-based, shot-based, and sequential comparison-based methods, will cause redundancy, i. e., similar frames in different cycles are extracted as key-frames. Methods in which other information, such as audio stream and objects in each frame are utilized for key-frame detection, are obviously not suitable for key-frame extraction from motions, as the only information available is the posture sequence in each motion.

A promising way is clustering-based extraction[26-28]. Particularly, similar frames are clustered into the same cluster, and a representative frame from each cluster is selected as a key-frame. We adopt the adaptive clustering-based key-frame extraction technique, in which similar frames in different cycles can be clustered into the same cluster. Define the similarity between two frames as function *Sim* about the weighted distance between them:

$$Sim(\boldsymbol{F}_1, \boldsymbol{F}_2) = f_d\left[\sum_k w_k (F_{1k} - F_{2k})^2\right]^{1/2} \tag{6-10}$$

where F_{1k} and F_{2k} are the motion parameters of frame $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$ respectively, w_k is the weight which indicating the significance of joint k. If these joints are from different levels in the hierarchy illustrated in Fig. 6. 24, we give higher weights to joints at higher levels whereas lower weights to those at lower levels empirically. Let $\boldsymbol{\delta}_i$ be the ith cluster, the clustering algorithm can be summarized as follows.

Step 1 Initialization: $\boldsymbol{F}_1 \rightarrow \boldsymbol{\delta}_1$, $\boldsymbol{F}_1 \rightarrow \boldsymbol{F}_{c1}$, (the centroid of $\boldsymbol{\delta}_1$), $1 \rightarrow$ *numCluster*.

Step 2 Get the next frame $\boldsymbol{F}_i$. If the frame pool is empty, end.

Step 3 Calculate the similarity between $\boldsymbol{F}_i$ and the centroid of an existing cluster $\boldsymbol{\delta}_k (k=1,2,\cdots,numCluster)$ according to equation (6-10). As $Sim(\boldsymbol{F}_i, \boldsymbol{F}_{ck})$.

Step 4 Determine the cluster closest to $\boldsymbol{F}_i$ through calculating *MaxSim* as follows:

$$MaxSim = \max_{k=1}^{numCluster} Sim(\boldsymbol{F}_i, \boldsymbol{F}_{ck})$$

If *MaxSim* is below a given threshold, it means $\boldsymbol{F}_i$ is not close enough to be put into any cluster, goto Step 5; otherwise put $\boldsymbol{F}_i$ into the cluster with *MaxSim*, and goto Step 6.

Step 5 $numCluster = numCluster + 1$. A new cluster is created: $\boldsymbol{F}_i \rightarrow \boldsymbol{\delta}_{numCluster}$.

Step 6 Update the cluster centroid as follows:

$$\boldsymbol{F}_{ck}=\frac{D\boldsymbol{F}'_{ck}+\boldsymbol{F}_i}{D+1}$$

where $\boldsymbol{F}'_{ck}$ and $\boldsymbol{F}_{ck}$ are the centroids before and after update, respectively, and D is the size of the old cluster. Go to Step 2.

According to Zhuang, et al.[29], the frame that is closest to the centroid of a cluster is selected as a key-frame. However, as the order among frames is lost during clustering, the extracted key-frame sequence is not consistent with the original temporal sequence. To preserve this essential attribute, considering frames in the same cluster are extracted sequentially, the first frame in each cluster is selected as the key-frame.

6.3.1.3 Motion Match

Most similarity measures for motions are defined based on their corresponding key-frame sequences. A simple method is the Nearest Center (NC) algorithm. Zhang, et al.[30] proposed a Nearest Neighbor (NN) approach, in which the similarity is defined as the sum of the most similar pairs of key-frames. As proposed by Zhao, et al.[31], these methods neglect the temporal order of frames. To address the problem, they presented a Nearest Feature Line-based method. Though this method accommodated the temporal correlation between key-frames, it could only be applied to retrieving motions using a single frame as the query sample.

Elastic match performed with a dynamic time warping algorithm is a nonlinear match method originally used in speech recognition and it has been successfully applied to online signature verification[32]. Elastic match can be used in comparison of all kinds of continuous functions about a continuous parameter, which is time typically.

Given two motions, $\boldsymbol{M}_1=\{\boldsymbol{F}1_1,\boldsymbol{F}1_2,\cdots,\boldsymbol{F}1_N\}$ and $\boldsymbol{M}_2=\{\boldsymbol{F}2_1,\boldsymbol{F}2_2,\cdots,\boldsymbol{F}2_M\}$, the distance D between them is defined as follows:

$$D=\frac{1}{2}\Big(\min_{\{\omega_1(i)\}}\sum_{i=1}^{N}d(i,\omega_1(i))+\min_{\{\omega_2(i)\}}\sum_{i=1}^{M}d(i,\omega_2(i))\Big) \tag{6-11}$$

where the first factor on the right part is the distance between $\boldsymbol{M}_1$ and the motion resulting from warping $\boldsymbol{M}_2$ according to the path defined by a time warping path $\omega_1(i)$, and the second factor on the right is the distance between $\boldsymbol{M}_2$ and the motion resulting from warping $\boldsymbol{M}_1$ according to the path defined by $\omega_2(i)$. $d(i,j)$ is the distance between the ith frame of $\boldsymbol{M}_1$ and the jth frame of $\boldsymbol{M}_2$, defined as:

$$d(i,j)=\sqrt{\sum_k w_k(\boldsymbol{F}1_{ik}-\boldsymbol{F}2_{jk})^2}$$

where $\boldsymbol{F}1_{ik}$ and $\boldsymbol{F}2_{jk}$ are the kth motion parameters in frame $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$, respectively, and w_k is the weight.

Let the right half part of D be DA, defined as:

$$DA = \min_{\{\omega_{1(i)}\}} \sum_{i=1}^{N} d(i, \omega_1(i)) .$$

Now let us take DA for example to solve the problem. The time warping path, for example, $\omega_1(i)$, is constrained by the following boundary and continuity conditions.

The boundary conditions ensure that the first and last frames of $\boldsymbol{M}_1$ are matched to the frame b and frame e of $\boldsymbol{M}_2$: $\omega_1(1)=b$, $\omega_1(N)=e$,

$$b=\min_{i \leqslant M} \arg\ (d(1,i) \leqslant \text{threshold})$$

where

$$e=\max_{i \leqslant M} \arg\ (d(N,i) \leqslant \text{threshold})$$

The continuity conditions restrict the match of the intermediate frames, and $\omega_1(i)$ is defined as a monotonically increasing function and thus the temporal order is preserved during match. We solve it recursively by applying dynamic programming in the following way[33]:

$$DA(i,j)=d(i,j)+\min\{DA(i-1,j), DA(i-1,j-1),$$
$$DA(i-1,j-2)\}$$

$$DA(1,b)=d(1,b)$$

where $DA(i-1,j-2)$ corresponds to skipping the $(j-1)$th frame of $\boldsymbol{M}_2$ and $DA(i-1,j)$ means that at least two frames of $\boldsymbol{M}_1$ correspond to the jth frame of $\boldsymbol{M}_2$.

6.3.2 Content-based Motion Retrieval

By now, the motion index tree has been constructed. We describe motion retrieval using the motion index tree in this section.

We devise a two-stage process to retrieve similar motions to a query sample $\boldsymbol{M}$. Determine a subset that contains promising similar motions to $\boldsymbol{M}$ and calculate distances/similarities between $\boldsymbol{M}$ and motions in the subset. The process of motion retrieval is outlined as follows.

Step 1 Fetch $\boldsymbol{M}_1$, the motion of the first-level joints, from the sample $\boldsymbol{M}$.

Step 2 Extract key-frame sequences from $\boldsymbol{M}_1$ to each motion in the sample set of Root in the motion index tree using the key-frame extraction described in Sect. 6.3.1, and calculate the distances between them using elastic match. Then get the K-nearest neighbors. Let $k_1, k_2, \cdots, k_c$ be the number of the nearest neighbors belonging to the child nodes of Root: $\omega_1, \omega_2, \cdots, \omega_c$, respectively.

The decision-making function is defined as $g_i(\boldsymbol{M})=k_i, i=1,2, \cdots, c$ and the decision-making rule is defined as follows:

If $g_j(\boldsymbol{M})=\max\limits_i k_i (i=1,2,\cdots,c)$, then $\boldsymbol{M}_1$ belongs $\boldsymbol{\omega}_j$, the jth child node of Root.

Step 3 Continue classification until a leaf node of the motion index tree is reached. Calculate the similarity between ***M*** and each motion stored in the subset in the leaf node, and select appropriate motions as the result according to the similarity.

6.3.3 Results and Discussions

To validate the effectiveness and efficiency of the proposed technique, we develop a prototype system of Content-based Motion Retrieval (CBMR) and test it on a motion library consisting of about 450 different motions. The composition of the library is shown in Table 6.3. Detailed discuss is as follows.

Table 6.3 Composition of the motion library

Motion class	Size of class	Motion class	Size of class
Walk forward	70	Rotate	32
Run forward	62	Run backward	32
Walk backward	54	Climb	28
Jump	48	Box	18
Squat	45	Wave hand	12
Ballet dancing	36	Fall	12

6.3.3.1 Key-frame Extraction

Both the adaptive clustering based key-frame extraction algorithm and the sequential comparison based method, in which a frame is selected as a new key-frame when the difference between this frame and the last key-frame is significant, are implemented and tested. For convenience, these two algorithms are called ACE and CE, respectively. In both algorithms, the similarity between two frames is defined as the reciprocal of the weighted distance between them.

First, we carry out experiment on two non-periodical motions: jump-kick and dive. Jump-kick consists of 70 frames and dive consists of 50 frames. The *MaxSim* value in ACE and the similarity in CE are shown in Fig. 6.25. For ACE, when the *MaxSim* value is below a given threshold, a new cluster is found and the current frame is selected as a new key-frame. From Fig. 6.25, it can be seen that totally 11 key-frames, including the first frame and the other 10 new key-frames, are extracted from jump-kick, and totally 10 key-frames, including the first frame and the other 9 new key-frames, are extracted from dive. For CE, the frame with its similarity below a given threshold is selected as a new key-frame. From Fig. 6.25, we can see that totally 16 key-frames, including the first frame and the other 15 new key-frames, are extracted from jump-kick, and totally 14 key-frames, including the first frame and the other 13 new key-frames, are extracted from dive. The extracted key-frame sequences are

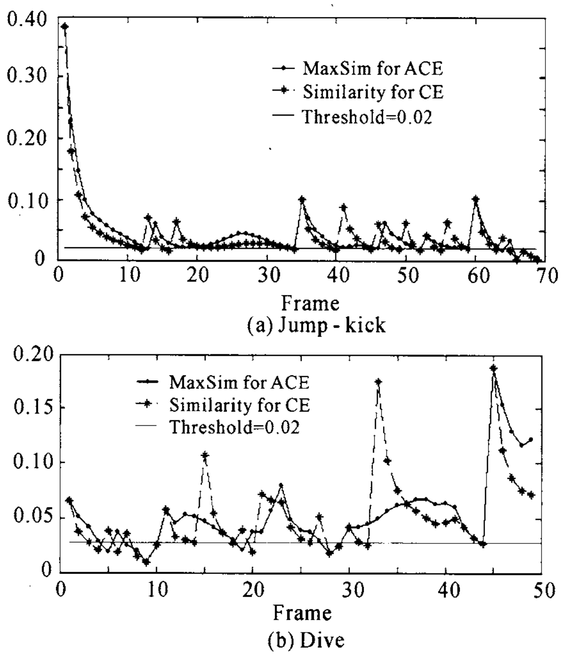


Fig. 6. 25 Extract key-frame sequences from non-periodical motions using ACE and CE

shown in Fig. 6. 26. As shown in Fig. 6. 25, more key-frames are extracted using CE than using ACE from both motions given the same threshold though the same similarity measure is adopted. Two reasons account for it. The maximum value in ACE is defined as the maximal similarity between the current frame and the centroid of each cluster, which means that the current frame is compared with the centroids of all the previously extracted clusters. In CE, the current frame is only compared with the most recent key-frame, which rules out the possibility for the current frame to match the most similar key-frame previously extracted. Another reason is that each motion is continuous, so the frames nearby are usually classified into the same cluster. Then the similarity between the current frame and the centroid decreases more slowly in ACE than that between the current frame and the last extracted key-frame in CE. As shown in Fig. 6. 27, the similar key-frame sequences can be extracted by ACE at a relatively high threshold and by CE at a relatively low threshold. These results show that the performances of CE and ACE are similar for non-periodical motions.

Second, two periodical motions, power-walk and hurdle, are tested on. Power-walk consists of 120 frames, and hurdle consists of 195 frames. Power-walk is composed of about 4 cycles of steps. From Fig. 6. 28(a), we can see that using ACE, five clusters are formed in power-walk, including the initial one and the appended 4. The first frame of each cluster is extracted as a key-frame. All these key-frames are extracted from the first 30

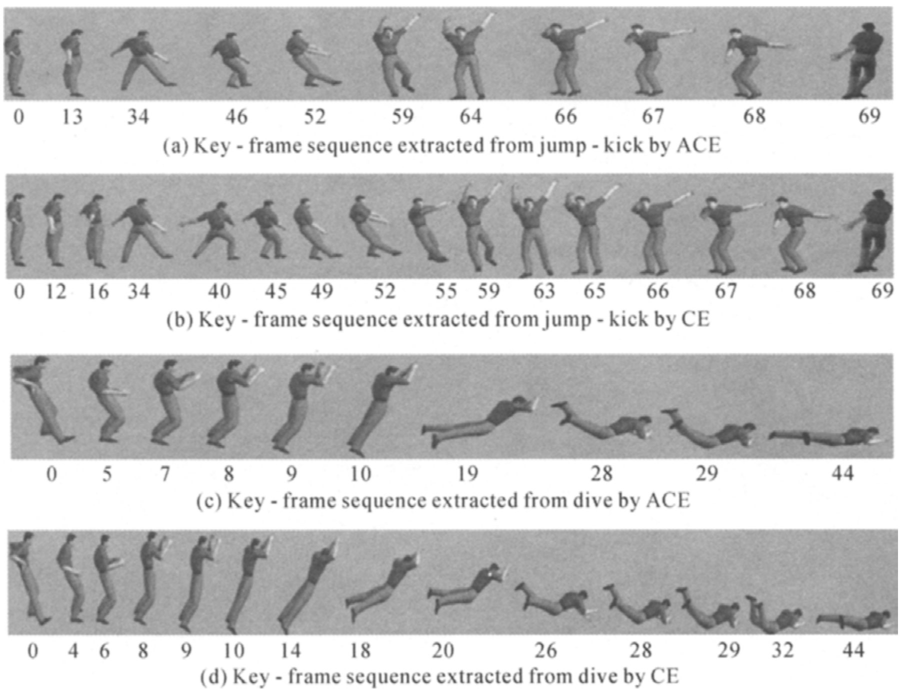

0 13 34 46 52 59 64 66 67 68 69
(a) Key - frame sequence extracted from jump - kick by ACE
0 12 16 34 40 45 49 52 55 59 63 65 66 67 68 69
(b) Key - frame sequence extracted from jump - kick by CE
0 5 7 8 9 10 19 28 29 44
(c) Key - frame sequence extracted from dive by ACE
0 4 6 8 9 10 14 18 20 26 28 29 32 44
(d) Key - frame sequence extracted from dive by CE

Fig. 6. 26 Key-frame sequences extracted from non-periodical motions

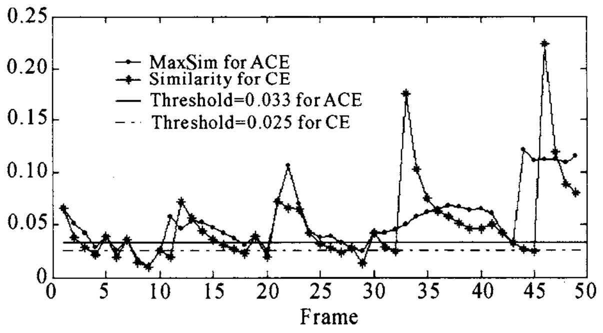


Fig. 6. 27 Key-frame extraction using ACE and CE with different thresholds. The key-frame sequence extracted from dive by ACE with threshold 0. 033 is 0 4 6 8 9 10 18 20 27 28 29 43, and the key-frame sequence extracted by CE with threshold 0. 025 is 0 4 6 8 9 11 18 20 27 29 32 45

frames, which compose the first cycle of power-walk. No frames in the successive cycles are extracted as key-frames. This shows the compactness of the extracted key-frame sequence. When it comes to CE, totally 29 key-frames, including the initial frame and the appended 28, are extracted from power-walk. From Fig. 6. 28(a), we can see that the last 28 key-frames can nearly be divided into 4 cycles and the last 3 seven-frame sequences are

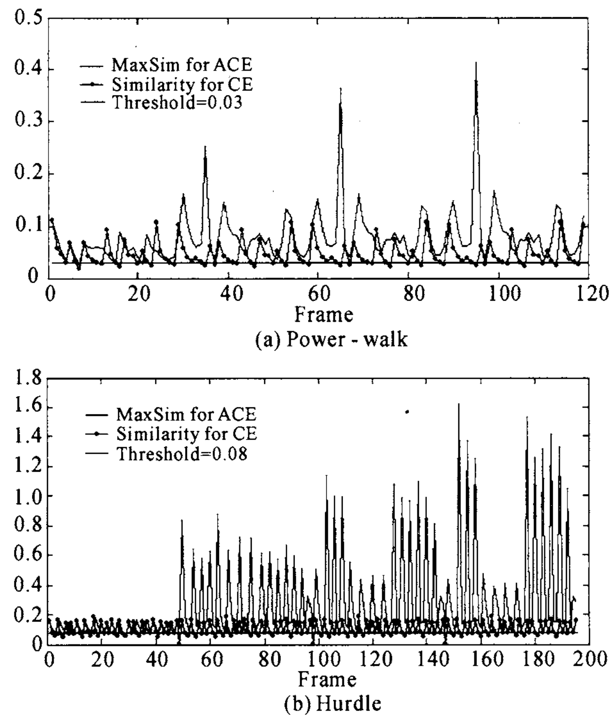


Fig. 6. 28 Extract key-frame sequences from periodical motions using ACE and CE

almost the repetitions of the first seven-frame sequence.

The experiment on hurdle shows similar result. All these demonstrate that ACE is superior to CE when applying to periodical motions.

6.3.3.2 Motion Match

To evaluate the performance of the elastic match algorithm, we implement this algorithm, as well as other methods, such as the NC method and the NN method, in our CBMR system. Some motions are selected from the library shown in Table 6. 3 as samples for queries to evaluate each algorithm. The distribution of these samples is stated in Table 6. 4. Based on these queries, we plot the average precision-recall graph for each algorithm as illustrated in Fig. 6. 29, from which we can see that elastic match is superior to both NC and NN. The main reason is that both NC and NN neglect the temporal order among the frame sequence of a motion. For example, walk cannot be distinguished from run, and walk-back cannot be distinguished from walk-forward either. Contrary to NC and NN, the continuity of the motion is preserved during the process of elastic match as a result of the constraint of continuity as addressed in Sect. 6. 3. 1. 1 The result of one query is shown in Fig. 6. 30. Each row represents a motion. The top one is the sample. The retrieved motions are sorted by similarity from top to bottom.

Table 6.4 Number of samples from each class is nearly proportional to the size of this class

Motion class	Number of samples	Motion class	Number of samples
Walk forward	6	Rotate	3
Run forward	5	Run backward	3
Walk backward	5	Climb	2
Jump	4	Box	2
Squat	4	Wave hand	1
Ballet dancing	3	Fall	1

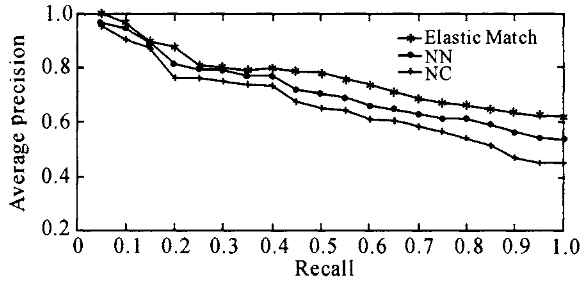


Fig. 6.29 Performance of elastic match

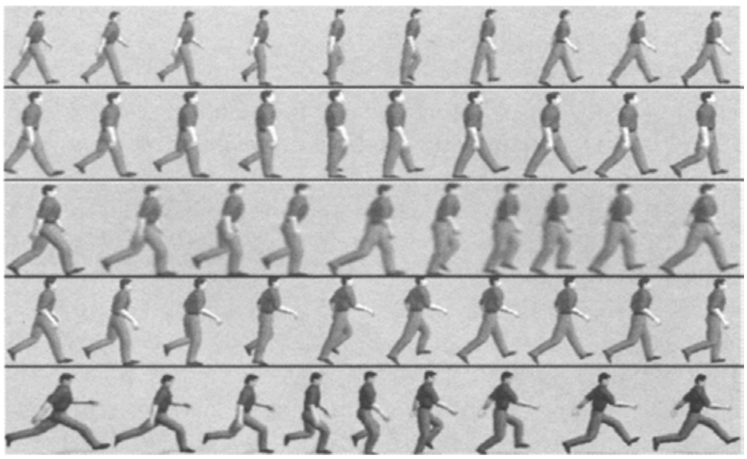


Fig. 6.30 Retrieval result of walk-forward

The calculation overhead of elastic match using dynamic programming is $O(mn)$, where m and n are the lengths of the two compared key-frame sequences, respectively. When retrieving a similar motion, the sample is firstly classified into the promising sub-library. The calculation overhead for the classification is $O(hsmn)$, where h is the height of the index tree and s is the size of the sample set in each non-leaf node. After classification, only the similarities between the samples and motions from the very sub-library are needed to be calculated and thus the efficiency is well improved.

In this section, a content-based 3D motion retrieval algorithm is proposed. The main contribution is the motion index tree constructed based on the hierarchical motion description. This motion index tree features the hierarchical effect that each joint has on the full-body motion, and serves as a classifier to determine a sub-library that contains promising similar motions to the query sample. Thus the efficiency can be well improved. We adopt nearest neighbor rule-based dynamic clustering algorithm to partition the motion library and construct the motion index tree. We employ a novel elastic match algorithm to calculate the similarity between two motions. The elastic match algorithm combines the dynamic time warping and dynamic programming, and has excellent performance on comparing two sequences. To improve the efficiency, we adopt an adaptive clustering-based key-frame extraction algorithm, which is especially competent for key-frame extraction from periodical motions.

References

1. Brand ME, Kettnaker V. Discovery and segmentation of activities in video. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8): 844-851, 2000.
2. Bradski G, Davis J. Motion segmentation and pose recognition with motion history gradients. International Journal of Machine Vision and Applications, 13(3): 174-184, 2002.
3. Arikan O, Forsyth DA, O'Brien J. Motion synthesis from annotations. ACM Transactions on Graphics (SIGGRAPH'03), 33(3): 402-408, 2003.
4. Kahol K, Tripathi P, Panchanathan S, Rikakis T. Gesture segmentation in complex motion sequences. In: ICIP'03, pp. II: 105-108, IEEE Computer Society, 2003.
5. Kahol K, Tripathi P, Panchanathan S. Automated gesture segmentation from dance sequences. In: AFGR'04, pp883-888, IEEE Computer Society, 2004.
6. Lu C, Ferrier JN. Repetitive motion analysis: segmentation and event classification. IEEE Transactions of Pattern Analysis and Meachine Intelligence, 26(2): 258-263, 2004.
7. Fod A, Mataric MJ, Jenkins OC. Automated derivation of primitives for movement classification. Autonomous Robots, 12(1): 39-54, 2002.
8. Pomplun M, Mataric MJ. Evaluation metrics and results of human arm movement imitation. In: Humanoids-2000, IEEE Computer Soci-

ety, 2000.

9. Wang T, Shum H, Xu Y, Zheng N. Unsupervised analysis of human gestures. In: IEEE Pacific Rim Conference on Multimedia, pp. 174-181, IEEE Computer society, 2001.
10. Barbic J, Safonova A, Pan J, Faloutsos C, Hodgins JK, Pollard NC. Segmenting motion capture data into distinct behaviors. In: GI'04, pp. 185-194, Canadian Human-Computer Communications Society School of Computer Science, 2004.
11. Tenenbaum JB, de Silva V, Langford JC. A global geometric framework for nonlinear dimensionality reduction. Science, 290(5500): 2319-2323, 2000.
12. Spath H. Cluster dissection and analysis: theory, FORTRAN programs, examples. pp. 226, Halsted Press, 1985.
13. Pickering MJ, Ruger S. Evaluation of key frame-based retrieval techniques for video. Computer Vision and Image Understand, 92 (2): 216-235, 2003.
14. Zhuang Y, Rui Y, Huang TS, Mehrotra S. Adaptive key frame extraction using unsupervised clustering. In: ICIP'98, IEEE Computer Society, 1998.
15. Zhuang Y, Pan Y, Wu F. Web-based multimedia information analysis and retrieval. Tsinghua University Press, 2002.
16. Liu F, Zhuang, Wu F, Pan Y. 3D motion retrieval with motion index tree. Computer Vision and Image Understanding, 92(2-3): 265-284, 2003.
17. Shen J, Sun S, Pan Y. Key-frame extraction from motion capture data. Journal of Computer-Aided Design and Computer Graphics, 16 (5): 719-723, 2004.
18. Lim IS, Thalmann D. Key-posture extraction out of human motion data by curve simplification. In: EMBC'01, pp. 1166-1169, IEEE Computer Society, 2001.
19. Lee J, Chai J, Reitsma PSA, Hodgins JK, Pollard NS. Interactive control of avatars animated with human motion data. In: SIGGRAPH'02, pp. 491-500, ACM Press, 2002.
20. Chui C, Chao S, Wu M, Yang S, Lin H. Content-based retrieval for human motion data. Journal of Visual Communication and Image Representation, 16(3): 446-466, 2004.
21. Muller M, Roder T, Clausen M. Efficient content-based retrieval of motion capture data. ACM Transactions on Graphics (SIGGRAPH'05), 24(3): 676-685, 2005.
22. Rosin PL. Techniques for assessing polygonal approximations of curves. IEEE Transactions on Pattern Recognition and Machine Intelligence, 19(6): 659-666, 1997.
23. Fritsch FN, Carlson RE. Monotone piecewise cubic interpolation.

SIAM Journal on Numerical Analysis, 17(2): 238-246, 1980.

24. Pullen K, Bregler C. Motion capture assisted animation: texturing and synthesis. In: SIGGRAPH ' 02, pp. 501-508, ACM Press, 2002.
25. Bian Z, Zhang C, Zhang X. Pattern recognition. pp. 241-244, Tsinghua University Press, 1999.
26. Kim C, Huang JN. Object-based video abstraction using cluster analysis. In: ICIP'01, pp. 657-660, IEEE Computer Society, 2001.
27. Ratakonda K, Sezan MI, Crinon R. Hierarchical video summarization. SPIE, 3653: 1531-1541, 1999.
28. Doulamis AD, Doulamis ND, Kollias SD. A fuzzy video content representation for video summarization and content-based retrieval. Signal Process, 80(6): 1049-1067, 2000.
29. Zhuang Y, Rui Y, Huang TS, Mchrotra S. Adaptive key frame extraction using unsupervised clustering. In: ICIP' 98, pp. 886-870, IEEE Computer Society, 1998.
30. Zhang H, Wu J, Zhong D, Smoliar SW. An integrated system for content-based video retrieval and browsing. Pattern Recognition, 30 (4): 643-658, 1997.
31. Zhao L, Qi W, Li SZ, Yang S, Zhang H. Key-frame extraction and shot retrieval using nearest feature line (NFL). In: Proceedings of International Workshop on Multimedia Information Retrieval, in Conjunction with ACM Multimedia Conference 2000, pp. 217-220, ACM Press, 2000.
32. Martens R, Claesen L. Dynamic programming optimization for on-line signature verification. In: International Conference on Document Analysis and Recognition, pp. 653-656, IEEE Computer Society, 1997.
33. Tappert CC. Adaptive on-line handwriting recognition. In: International Conference on Pattern Recognition, pp. 1004-1007, IEEE Computer Society, 1984.

7

Intelligent Motion Data Reusing Techniques

In order to reuse the existing 3D human motion data to create animation and improve the efficiency of animation creation, the motion reusing technique is becoming a hotspot in research. Generally, the motion reusing includes motion editing, motion synthesis and motion retargeting, etc.

The intelligent motion data reusing techniques are different from traditional ones, which need a great deal of laborious work from animators. It emphasizes on the process of reusing motion data automatically and intelligently. This chapter is based on author's research on motion data reusing in these years and will illustrate in detail the motion editing and synthesizing based on wavelet transform, the motion graph model based on Markov chain and the automatic editing and synthesizing in human motion style.

7.1 3D Motion Editing and Synthesis Based on Wavelet Transform

Motion Capture (MoCap) data editing is critical in creating complicated animation and improving the reusability of motion data. This section presents a 3D computer animation technique, which can control the motion details and deal with the motion abstractly. This technique manages the motion data both in frequency and spatio-temporal domain. First of all, it adopts wavelet transform to analyze the motion signal in multi-resolution, and realizes the motion feature enhancement, motion blending and motion feature extraction, etc. Then according to the motion feature and animator's requirement, the spatio-temporal constraints are added into the motion and the objective function is built and resolved to retain motion reality.

7.1.1 Hierarchical Motion Description

The human model used in the research is shown in Fig. 7. 1, which includes 16 joints. By using MoCap system the positions of these 16 joints

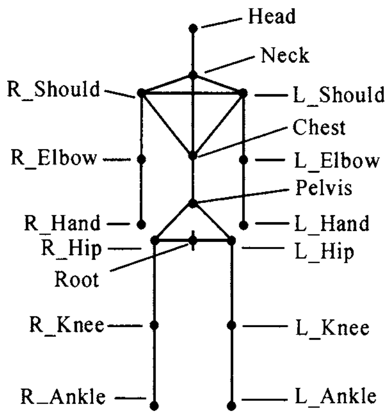


Fig. 7. 1 Human model

$$\begin{tikzcd}
 & & \text{Root} \arrow[dll] \arrow[d] \arrow[drr] & & \\
\text{R\_Hip} \arrow[d] & & \text{Pelvis} \arrow[d] & & \text{L\_Hip} \arrow[d] \\
\text{R\_Knee} \arrow[d] & & \text{Chest} \arrow[dl] \arrow[d] \arrow[dr] & & \text{L\_Knee} \arrow[d] \\
\text{R\_Ankle} & \text{R\_Should} \arrow[d] & \text{Neck} \arrow[d] & \text{L\_Should} \arrow[d] & \text{L\_Ankle} \\
 & \text{R\_Elbow} \arrow[d] & \text{Head} & \text{L\_Elbow} \arrow[d] & \\
 & \text{R\_Hand} & & \text{L\_Hand} &
\end{tikzcd}$$

Fig. 7. 2 Hierarchical motion describing

can be obtained. Using joints' positions to describe motion is inconvenient in utilizing these data, since joints' motions are independent by using this kind of description. In our research, the human motion is described hierarchically by a tree model (Fig. 7. 2)[1]. The root of the tree corresponds to the root joints in human model and other nodes correspond to other joints. The motion composed of translation and rotation includes root node's translation and the lower node' rotation according to the upper node. The root rotation determines human model's orientation and other nodes are rotated in the parents' coordinating systems, whose origins are their corresponding upper nodes. The joints' positions can be calculated by the length of bone and rotation vector. For example, in Fig. 7. 1 the right ankle, whose position and rotation vector is:

$$\boldsymbol{P}^{\mathrm{W}}_{\mathrm{R\_Ankle}}(x,y,z)=\boldsymbol{T}_{\mathrm{Root}}\boldsymbol{R}_{\mathrm{Root}}\boldsymbol{T}_{\mathrm{R\_Hip}}\boldsymbol{R}_{\mathrm{R\_Hip}}\boldsymbol{T}_{\mathrm{R\_Knee}}\boldsymbol{R}_{\mathrm{R\_Knee}}\boldsymbol{T}_{\mathrm{R\_Ankle}}\boldsymbol{R}_{\mathrm{R\_Ankle}}\boldsymbol{P}^{\mathrm{L}}_{\mathrm{R\_Ankle}}(x,y,z) \tag{7-1}$$

here $\boldsymbol{P}^{\mathrm{W}}_{\mathrm{R\_Ankle}}(x, y, z)$ represents right angle's world coordinate and $\boldsymbol{P}^{\mathrm{L}}_{\mathrm{R_Ankle}}(x,y,z)$represents right angle's initial coordinate in right knee's coordinate system. $\boldsymbol{T}_l$ is a vector translating joint l from current coordinate system to its parent's coordinate system. $\boldsymbol{R}_l$ composed of three rotation vectors according to x, y, z axis, rotates the joint l around its parent.

After calculating the rotation vector according to equation (7-1), the human motion can be described as:

$$\boldsymbol{M}(t)=(\boldsymbol{T}^{\mathrm{W}}_{\mathrm{Root}}(t),\boldsymbol{R}_1(t),\cdots,\boldsymbol{R}_l(t),\cdots,\boldsymbol{R}_n(t)) \qquad (l=1,2,\cdots,n)$$

where $\boldsymbol{T}^{\mathrm{W}}_{\mathrm{Root}}(t)$ is the world coordinate of Root and $\boldsymbol{R}_l(t)$ is the joint l's rotation vector at time t.

7. 1. 2 Motion Signal Analysis by Wavelet

The multi-resolution analysis of wavelet, which focalizes on any object's details, has satisfied localization features in both temporal and frequency domains, so the adoption of wavelet in motion editing can effectively control the motion details. The low frequency part in wavelet analysis represents the motion's elementary features, which can be effectively edited.

Supposing the dispersed value ($\boldsymbol{C}^0$) of motion signal $f(t)$ ($f \in \boldsymbol{L}^2(\mathbf{R})$) is:

$$\boldsymbol{C}^0 = \{\boldsymbol{C}_j^0\} \ (j, n \in \mathbf{Z},\ 0 \leqslant j < n,\ 2^m \leqslant n < 2^{m+1})$$

where n is the number of input motion's frames, m is the frequency band of motion signal. According to multi-resolution's definition:

$$\boldsymbol{V}_0 = \boldsymbol{V}_m \oplus \boldsymbol{W}_m \oplus \boldsymbol{W}_{m-1} \oplus \cdots \oplus \boldsymbol{W}_1 \tag{7-2}$$

If $f(t)$ is extended by equation (7-2) and according to wavelet transform:

$$f(t) = \sum_{k=-\infty}^{\infty} \sum_{j=1}^{m} d_{j,k} \boldsymbol{\Psi}_{j,k}(t) \sum_{k=-\infty}^{\infty} c_{m,k} \boldsymbol{\varphi}_{m,k}(t) \tag{7-3}$$

$$d_{j,k} = \langle f(t), \boldsymbol{\Psi}_{j,k}(t) \rangle \tag{7-4}$$

$$c_{m,k} = \langle f(t), \boldsymbol{\varphi}_{m,k}(t) \rangle \tag{7-5}$$

where $\boldsymbol{\varphi}(x)$ and $\boldsymbol{\Psi}(x)$ are corresponding scale function and wavelet function. Equations (7-4) and (7-5) can be adopted in motion signal analysis, while equation (7-3) is used in rebuilding motion signal.

7.1.3 3D Motion Analysis and Synthesis Based on Wavelet Transform

7.1.3.1 Motion Feature Enhancements

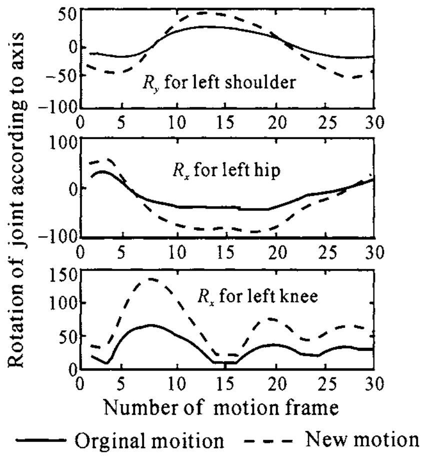
By analyzing the motion in frequency domain, the subparts in motion signal with different frequency bands represent motion features on different layers. The low frequency part represents motion's elementary features and the high frequency part represents motion's details. The motion features can be enhanced or weakened to create new motion by dealing with the signal in corresponding frequency band. For example, the enhancement of the high frequency part in the motion signal can strength some motion details. The motion feature enhancing algorithm is elaborated below.

The initial motion signal is supposed to be $f(t)$. According to equations (7-4) and (7-5), the motion signal is described by equation (7-3), so the enhancement of motion signal $\boldsymbol{g}(t)$ is:

$$\boldsymbol{g}(t) = \sum_{i=-\infty}^{\infty} \sum_{j=1}^{m} k_{j,i} d_{j,i} \boldsymbol{\Psi}_{j,i}(t) + \sum_{i=-\infty}^{\infty} r_i c_{m,i} \boldsymbol{\varphi}_{m,i}(t) \tag{7-6}$$

where $k_{j,i}$, r_i are weights used for motion signal enhancement.

Through such a gain controller in frequency band, the animator can edit the motion data. For example, given a series of normal walk (see Fig. 7. 3), coordinating motion signal's low frequency part of left (right) hip joint and left (right) knee joint can achieve the walk motion editing with



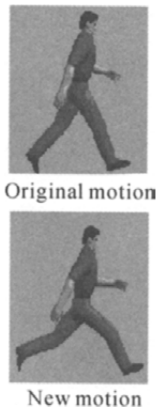


Fig. 7.3 Walking step enlargement

varied steps; the arm's motion with varied swing step can be obtained by coordinating the low frequency part of the left (right) shoulder's motion. By enhancing the high frequency part in left (right) knee's motion, the trembling details in walking process are stressed to obtain the jittering walking.

7.1.3.2 Motion Blending

The motion blending based on wavelet transform decomposes the input original motion data into varied frequency feature domains, and different weights are adopted to create new motion.

Before decomposing and blending the motion, the input motion signal must be processed synchronously. For example, the motion of walking and running must be coordinated synchronously. The time warping, which is commonly used in audio processing, is adopted here.

$$g(t)=f(h(t)) \tag{7-7}$$

in which the piecewise linear function $h(t)$ is time warping function; $f(t)$ is the original motion signal and $g(t)$ is the motion signal after using time warping. For any two motion signals, the animator will appoint several time corresponding points. One of them is assumed to be the reference signal. According to equation $h(t)$ and equation (7-7), the synchronization can be achieved between the two motion signals (shown in Fig. 7.4).

Since the motion data are discrete, there are two problems existing in the original motion signal sampling process: (1) when $h'(t)$ is less than 1, how to determine the motion information existing in the two frames in original motion; (2) how to solve the problem of insufficiency sampling problem. The first problem can be solved by linear interpolation or cubic interpolation, since in motion signals the time interval between two consecutive frames is short; for the second problem, the reference motion should be changed to diminish $h'(t)$. If there are only two motions, then $h'(t)=(h'_0(t))^{-1}<1$, in which $h_0(t)$ is the initial time warping function.

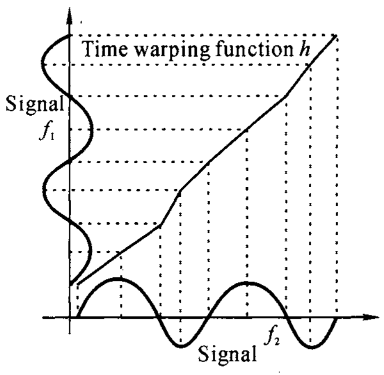


Fig. 7. 4 Time warping function. f_2 is the reference function; time warping function h achieves the synchronization of f_1 and f_2

The blending algorithm based on wavelet transform can be used in synthesizing new motion after constructing the synchronous relationship in the initial motion signals. The algorithm is described as follows.

Step 1 Decompose the signal of joint l, $f_{i,l}(t)$, in the ith motion signal by using wavelet analysis, then $d_{i,l,j,k}$ and $c_{i,l,j,k}$ $(1\leqslant j\leqslant m)$ can be obtained, m is number of layers in wavelet analysis.

$$\left.\begin{aligned} d_{i,l,j,k} &= \langle f_{i,l}(t), \boldsymbol{\Psi}_{j,k}(t)\rangle \\ c_{i,l,j,k} &= \langle f_{i,l}(t), \boldsymbol{\varphi}_{j,k}(t)\rangle \end{aligned}\right\} \tag{7-8}$$

Step 2 Apply the blending weights $\boldsymbol{\alpha}_{i,l,j} = \{\alpha_{i,l,j,k}\}_{k\in Z}$ in the motion signal's low frequency parts and $\boldsymbol{d}_{l,j,k}$ is calculated as:

$$\boldsymbol{d}_{l,j,k} = \sum_{i=1}^{n} \boldsymbol{\alpha}_{i,l,j,k}\, d_{i,l,j,k} \tag{7-9}$$

Step 3 Apply the blending weights $\boldsymbol{\beta}_{i,l,j} = \{\beta_{i,l,j,k}\}_{k\in Z}$ in the motion signal's high frequency parts and the synthesized high frequency part $\boldsymbol{c}_{l,j,k}$ is:

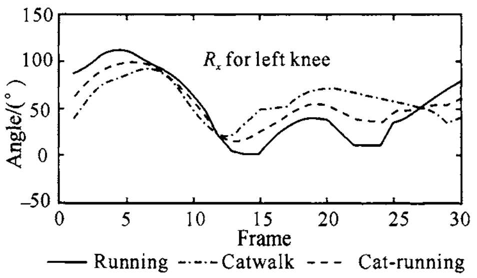
$$\boldsymbol{c}_{l,j,k} = \sum_{i=1}^{n} \boldsymbol{\beta}_{i,l,j,k}\, c_{i,l,j,k} \tag{7-10}$$

Step 4 Create new motion signal by reconstructing the wavelet analysis coefficients of the synthesized motion signal.

By adopting the above algorithm in synthesizing Catwalk and natural running, the Cat-running can be created by choosing appropriate blending weights (shown in Fig. 7. 5).

7.1.3.3 Motion Feature Extraction and Synthesis

By analyzing motion signal in frequency domain, we find that the signals with different frequency bands have different features, which can be extracted and synthesized with the main motion signal to create new motion. We adopt the high and low frequency complementary algorithm (see Fig. 7. 6) in motion feature extraction and synthesis. Firstly the algorithm decomposes



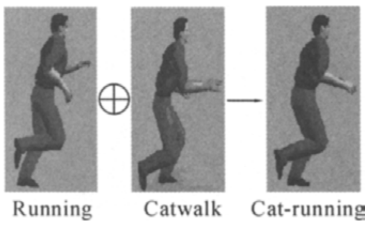


Fig. 7. 5 Blending of the running and Catwalk

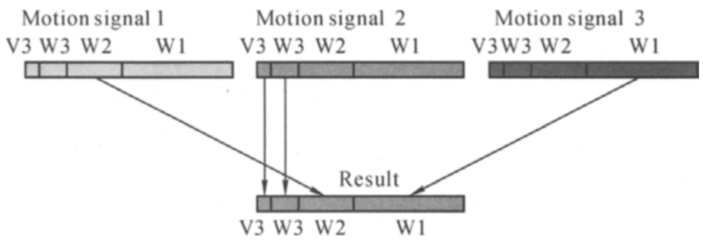


Fig. 7. 6 Complementary algorithm from high and low frequency domain

the input motion signals into approximate part and particular part. The approximate part, which is calculated by scale function having low pass filter features, reflects the original motion. The particular part is calculated by wavelet function and represents the information in original motion's certain frequency band. In motion feature extraction, the approximate part is chosen to reflect new motion's basic features, and the particular parts in other motions are chosen to reflect new motion's details.

Since the feature extracted from the signal is particular part, the original motion needs not to be similar. The motion synchronization can be achieved by using the time warping algorithm introduced in the last section.

In the following part a new motion called Cat-running is synthesized by extracting the detail features from Catwalk motion and the approximate part from running motion. This example shows how to extract the motion features and synthesize new motion by using the complementary algorithm.

Step 1 Running is determined to be the dominant motion.

Step 2 By analyzing the original motion, the Catwalk is reflected by the motion of left (right) hip.

Step 3 Synchronize the original motions by using time warping algorithm.

Step 4 Analyze the left (right) hip's motion signal by using wavelet. The approximate part in the dominant motion (running) is chosen to be the approximate part in the new motion, and the high frequency part in the lower frequency band is chosen to be part of the particular part. Other particular parts are chosen from Catwalk.

Step 5 Reconstruct the motion signal according to equation (7-3) and the new motion signal with Catwalk features can be created.

Step 6 The constructed left (right) hip's motion can be replaced into the left (right) hip's motion in the original running, so the Cat-running can be obtained (in Fig. 7. 7).

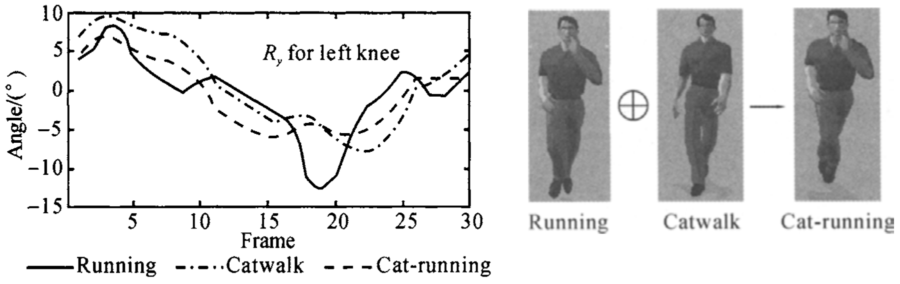


Fig. 7. 7 Feature extracting from Catwalk

7. 1. 4 Management of Motion Reality

The motion reality is guaranteed by a series of constraints which are embedded in the original motion data. Some motion editing operations will break several constraints and the manipulations in frequency domain cannot guarantee the reality. Since the motion is a series of changing postures according to time, some constraints are added into spatio-temporal domain in the editing process and the damages of the constraints supporting motion reality will be reduced.

We adopt the motion optimization algorithm based on spatio-temporal constraints [2, 3] to optimize the motion data in frequency domain, so the reality of the new motion is guaranteed. The algorithm is described bellow.

Step 1 Make sure a set of spatio-temporal constraints, and adopt the formalization method to define it.

Step 2 Define the object function for motion optimization.

Step 3 Adopt the Inverse Kinematics (IK) and Numerical Optimization Method to create new motion.

Step 4 If the request is not met, the spatio-temporal constraints or optimized object function are formalized again to create new motion.

7. 1. 4. 1 Spatio-temporal Constraints

The purpose of using spatio-temporal constraints is to preserve several inherent attributes and control the motion's variation. It can be described by a set of equations:

$$\boldsymbol{F}(t_c, m) = \boldsymbol{c} \tag{7-11}$$

where t_c represents the time of restriction and m is the motion parameters, which can be described by position coordinates or by rotation and translation, $\boldsymbol{c}$ is the matrix of scalar quantity, $\boldsymbol{F}(t_c, m)$ is the motion in corresponding frame.

The spatio-temporal constraints from virtual environment and animator are listed below.

- The constraints on the character:

$$M(t)=\begin{bmatrix} T(t) \\ R_i(t) \end{bmatrix}=\begin{bmatrix} C_0 \\ C_1 \end{bmatrix}$$

  in which $M(t)$ represents the parameter of rotation and translation, $T(t)$ is a vector representing the position of Root at time t, C_0 is the translation constant, $R_i(t)$ is the rotation vector of joint i at time t, C_1 is the rotation constant.
- The constraints on the character's supporting point: $p_i(t)=C_2$, in which $p_i(t)$ represents the position of joint i at time t, C_2 is the position constant.

7.1.4.2 Objective Function

The objective function regulates the entire motion. The function of energy consuming and the function of motion discrepancy before and after editing are assumed to be object functions. Since the objective function is described by motion parameters, which have different effects on postures, the parameters' sum of weighted square is adopted to describe the objective function. In our motion editing system, two kinds of objective functions are adopted. The first one is to minimize the errors summation of the joints before and after editing:

$$\min\left(sum=\int_{t_{\text{start}}}^{t_{\text{end}}}\sum_{k=1}^{16}\|p_k(t)-\hat{p}_k(t)\|^2\,\mathrm{d}t\right) \qquad (7\text{-}12)$$

where $p_k(t)$ is the kth joint's position vector after editing, $\hat{p}(t)$ is the kth joint's perfect position before editing, t_{start} and t_{end} represent motion editing interval's start and end point respectively.

The second one is to minimize the error summation of the motion posture before and after editing:

$$\min\left(sum=\int_{t_{\text{start}}}^{t_{\text{end}}}\left(w_1\|T(t)-\hat{T}(t)\|^2+w_2\sum_{k=1}^{k=16}\|R_k(t)-\hat{R}_k(t)\|^2\right)\mathrm{d}t\right) \qquad (7\text{-}13)$$

where $\hat{T}(t)$ and $T(t)$ represent the translation vector before and after editing respectively, $\hat{R}_k(t)$ and $R_k(t)$ represent the kth joint's rotation vector before and after editing respectively, w_1 and w_2 are the weighted values of the formalized translation vector and rotation vector.

7.1.4.3 Solving Method

In the motion editing method based on spatio-temporal constraints, the dynamics and IK are usually adopted to solve the spatio-temporal equation.

The dynamics determines character's every joint' position and orientation directly, while IK calculates the joints' position and orientation according to user's intervention. Comparing with dynamics, IK can reduce the laborious work, since the user only needs to designate the position of the end joint, and other joints' positions can be calculated by IK module automatically. Although IK approach can resolve all the results, its computation cost is huge. We use MoCap data as initial value and combine the IK approach and Numerical Optimization Method to resolve the spatio-temporal constraint problem.

7.1.4.4 An Example of Optimization Algorithm

As described in Sect. 7.1.3, the motion enhancement algorithm is adopted to create walking with varied steps from normal walking motion. These new motions have lost some kind of reality: (1) In some frames the foot gets onto ground shown in the 9th, 11th, 13th frames in Fig. 7.8(a), or two feet leave ground shown in the 1st, 3rd, 17th frames; (2) When the steps are increased, the "foot-skating" appears; whereas the "hurtling" appears when the steps are reduced.

Problem (1) happens, because the new motion breaks the constraint that in the walking at least one foot must stay on the ground; problem (2) happens, because new motion breaks the constraint that in the walking the footprint should not be changed. Because of these two problems, the constraints for the human motion are determined, like fixing $\boldsymbol{P}_{\mathrm{R\_Ankle}}(x,y,z)$, and according to equation (7-1) and IK approach, the $\boldsymbol{T}_{\mathrm{Root}}$ can be calculated. The satisfied motion gestures are shown in Fig. 7.8(b).

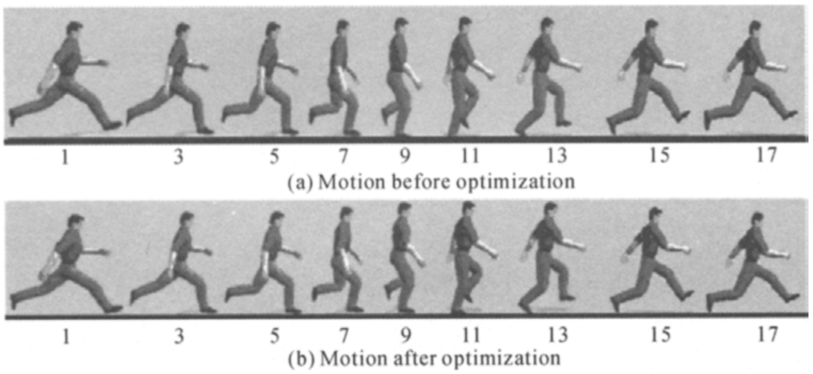


Fig. 7.8 Motion optimization in walking

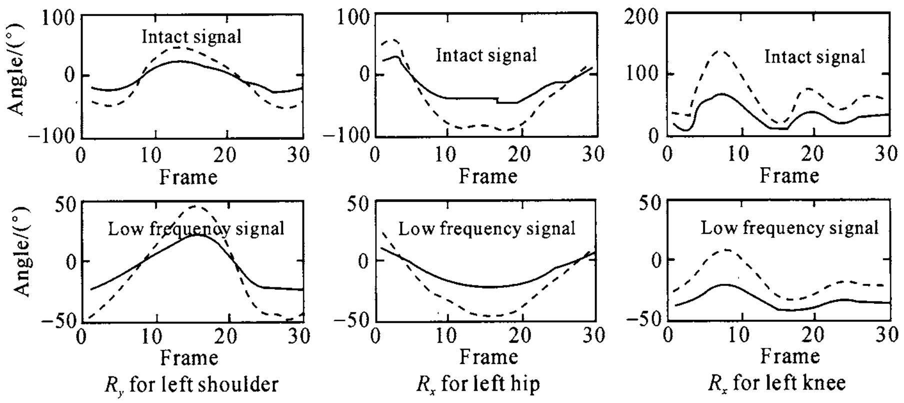
7.1.5 Results

7.1.5.1 Motion Enhancement

Inputting an adult's natural walking (Fig. 7.9(a)) and by enhancing the motion's left (right) hip, left (right) knee R_x and left (right) shoulder

R_y's low frequency signal, the step of the motion is enhanced. The natural walking with vigorous strides is shown in Fig. 7. 9(c). If the high frequency part is enhanced, the walker's nervousness can be expressed (Fig. 7. 10).

(a) Key-frames in natural walking



(b) Enhancement in low frequency signal

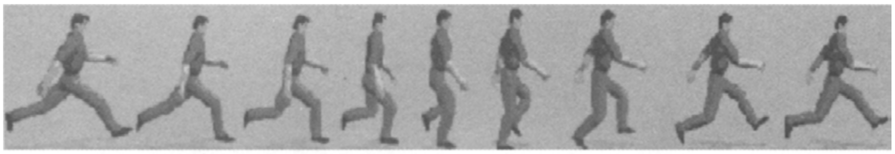
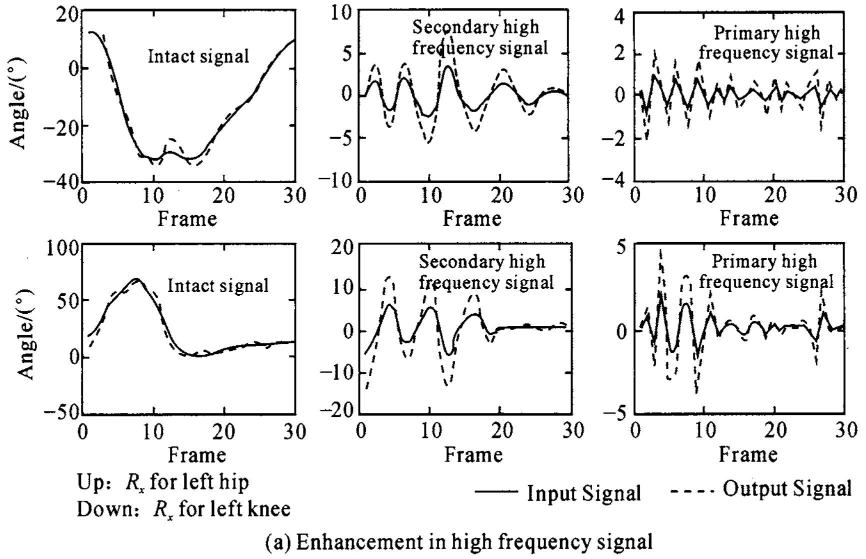

(c) Key- frames in the walking with vigorous strides

Fig. 7. 9 Steps increasing in motion enhancement

7.1.5.2 Motion Blending

The algorithm of motion blending can effectively synthesize the motion with different features to generate new motion retaining the original motion' s features. Since wavelet transform has satisfied performance for multi-resolution analysis, it can be adopted in varied feature domains. Different blending weights are utilized to generate new motion.

For example, a normal running (Fig. 7. 11(a)) is used as the initial input to blend with walking motions having varied styles (Fig. 7. 11(b), (d) and (f)). Fig. 7. 11(c), (e) and (g) show the blending results. It can be seen that not only character's gestures (Fig. 7. 11(b), 7. 11(d), (c), (e)) but also their emotions (7-11(f), (g)) are translated into the new motions.



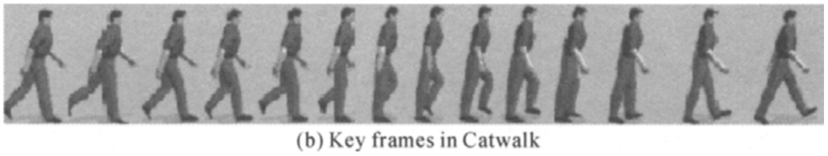


Fig. 7. 10 Trembling walk in motion enhancement

7.1.5.3 Motion Feature Extraction and Synthesis

In the algorithm, the specific features, which are reflected by motion signal's frequency domain, are firstly extracted to fuse with the designating dominant signals. For example, in Catwalk (shown in Fig. 7. 12(b)), the swift motion of thigh is extracted to fuse with running to produce a new motion (Fig. 7. 12(c)). Firstly the input motion signal is decomposed into 4 layers by using wavelet. By analyzing the motion signal (Fig. 7. 12(a)), the high frequency part D3, D2 and D1 reflect the features of Catwalk. They are combined with the corresponding joints' low frequency part A4 and D4 to create the new motion (see Fig. 7. 12(c)).

This section proposes a motion editing algorithm, which can manipulate the motion data both in frequency and spatio-temporal domains. Firstly wavelet transform is adopted to analyze the motion data. The algorithms of motion feature enhancement, motion blending and motion feature extraction and synthesis are achieved. Then according to the animator's requests, the spatio-temporal constraints are added into motion to retain the motion reality. The experimental results show that these algorithms can not only edit the motion abstractly and conveniently, but also retain the motion reality.

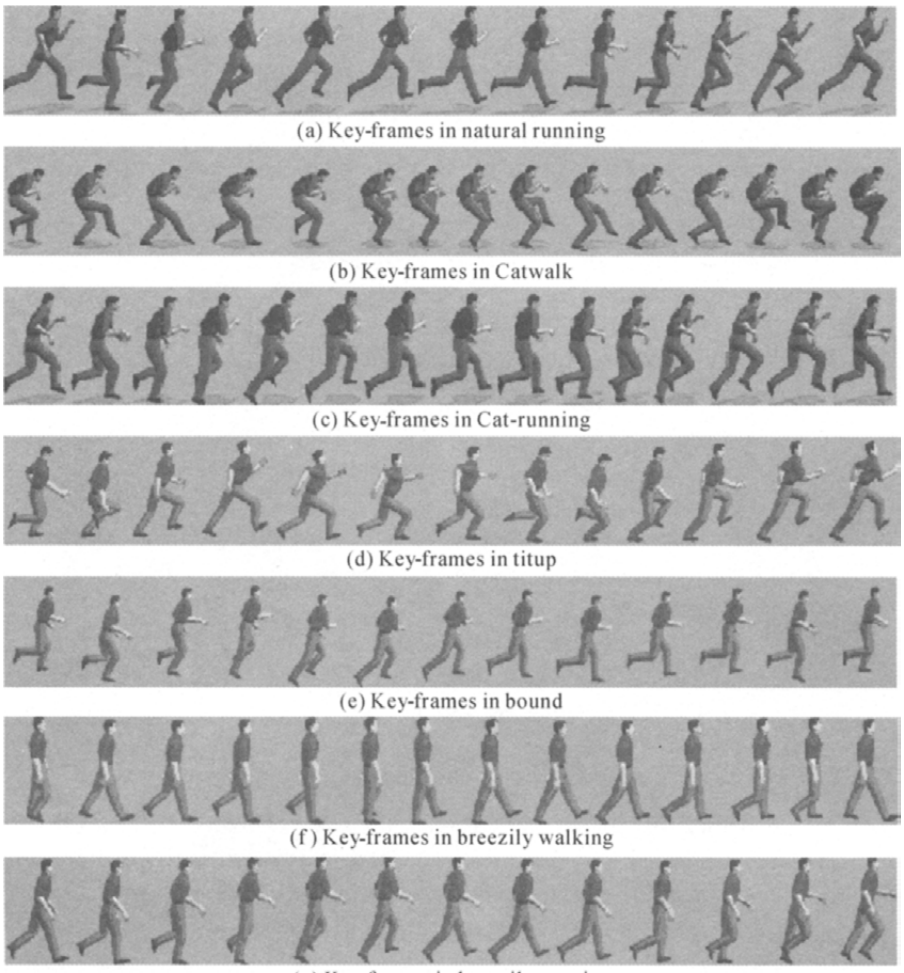
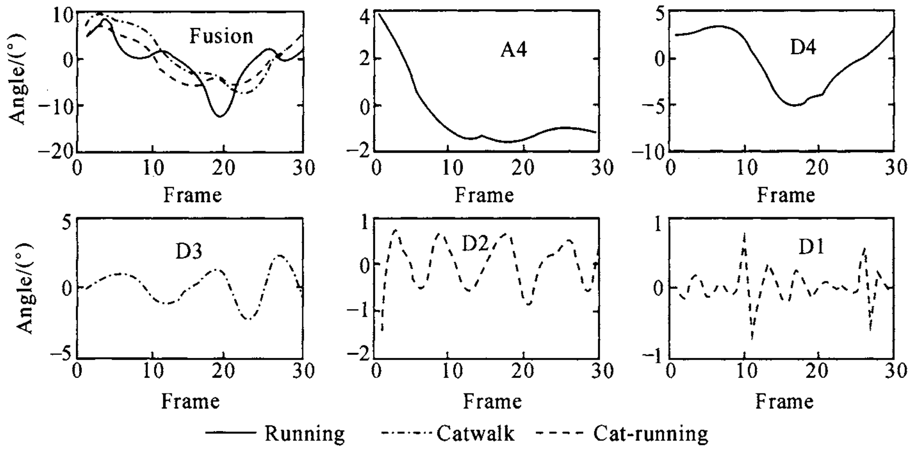

(a) Key-frames in natural running
(b) Key-frames in Catwalk
(c) Key-frames in Cat-running
(d) Key-frames in titup
(e) Key-frames in bound
(f) Key-frames in breezily walking
(g) Key-frames in breezily running

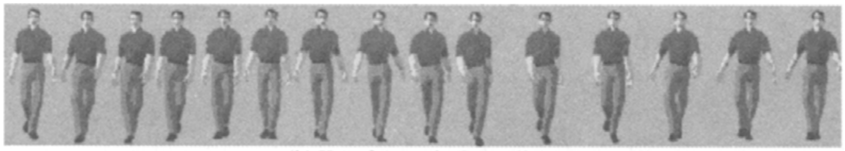
Fig. 7.11 Motion blending

7.2 Motion Graph Modeling Based on Markov Chain

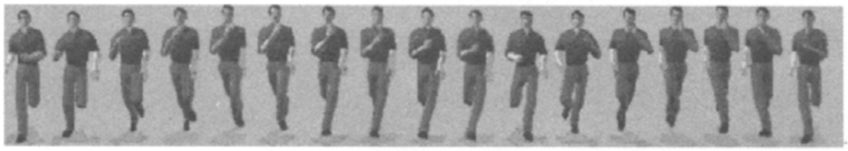
In this section a motion graph based motion synthesis framework is discussed in detail. First the motion units are picked up from motion database by using greedy algorithm. Each unit is defined to be a motion segment representing basic dynamics, and the units with similar dynamic features are grouped into one cluster. The motion is modeled to be a first-order Markov process and the motion clusters are used as the states in Markov chain. The connection between motion clusters are modeled by a directed graph



(a) The high and low frequency signals and the fusion signal using complementary alorithm



(b) Key-frames in Catwalk



(c) Key-frames in Cat-running

Fig. 7. 12 Motion extraction and synthesis for Catwalk

called motion graph. Finally a new motion can be created according to the motion graph in following two steps: (1) finding a path on the motion graph, which is composed of the motion units; (2) connecting these units smoothly. In this framework, two motion synthesis techniques are proposed: a series of similar but not the same motions are generated for the crowd animation by randomly motion sampling, and new motion can be generated along the synthesized motion path.

7. 2. 1 Motion Graph Building

The motion graph is a new motion describing method, which represents the connection of the motion segments in the library, as show in Fig. 7. 13. In the

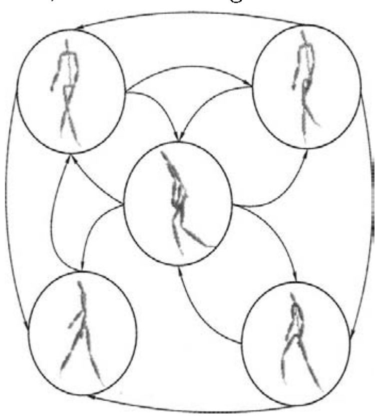


Fig. 7. 13 Motion graph

motion graph, each node represents a motion segment, which could be a frame or a series of frames; the directed edge *ij*'s weight means the connecting degree from *i* to *j*. In Arikan and Forsyth's [4] research, each node is a motion frame and the connecting degree is promoted a lot. But the weakness is that the constructed motion graph has so many vertexes and the new motion does not have smooth transitions between frames. If a long motion segment or the whole motion is used as the vertex, the realistic movement can be retained; however the motion graph's connectivity and underlying space for new motion synthesis are sacrificed.

Our strategy is to take the extracted motion cluster from MoCap data as the vertex in the motion graph, which is similar with Li's approach [5]. The motion units are kept in the motion cluster and the new motion is created from these units, so the new motion can perfectly retain the motion reality and enlarge the underlying motion synthesis space.

7.2.1.1 Extraction of Motion Unit and Motion Cluster

The motion unit is formalized to be a basic motion segment showing the motion's dynamic features, and can be described by a second-order dynamic motion model: $\boldsymbol{MD}=\{\boldsymbol{F}_0, \boldsymbol{A}, \boldsymbol{B}\}$, in which $\boldsymbol{F}_0$ is the initial frame, $\boldsymbol{A}$ and $\boldsymbol{B}$ are dynamic parameters. Each frame $\boldsymbol{F}_t$ in the motion unit can be approximately rebuilt by corresponding dynamics model:

$$\boldsymbol{F}_t=\boldsymbol{F}_0+\boldsymbol{A}t^2+\boldsymbol{B}t$$

The motion units having similar dynamic features are clustered into a group called motion cluster:

$$\boldsymbol{MC}=\{\boldsymbol{MD}, \boldsymbol{MP}_1, \boldsymbol{MP}_2, \cdots, \boldsymbol{MP}_N\}$$

where $\boldsymbol{MD}$ is a public dynamics model, $\boldsymbol{MP}_i$ is the motion unit in motion cluster $\boldsymbol{MC}$.

Here the algorithm is described in detail by using a character composing of a global position, orientation and other 17 joints. Each gesture (frame) can be formalized to be a vector having 57 dimensions, so the motion $\boldsymbol{M}$ having T frames can be represented by a $T\times 57$ matrix. In order to reduce the computing complexity, $\boldsymbol{M}$ is decomposed by Singular Value Decomposition (SVD) algorithm, and the dominant motion is extracted:

$$\boldsymbol{M}=\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}}$$

where $\boldsymbol{U}$ is a $T\times T$ matrix, $\boldsymbol{\Lambda}$ is a $T\times 57$ matrix, and $\boldsymbol{V}$ is a 57×57 matrix. Let $\boldsymbol{U}_{1-q}$ be a $T\times q$ matrix, the algorithm of extracting motion units and motion cluster from MoCap data is elaborated below.

Step 1 Pick the motion $\boldsymbol{M}$ from the library and apply SVD to get the matrix $\boldsymbol{U}_{1-q}$.

Step 2 Take the first $T_{\min}$ frames from $\boldsymbol{U}_{1-q}$ and form the motion segment

tMP_1, apply the least squares method to calculate the second-order dynamic model A_{1-q}, B_{1-q} and $F_{0,1-q}$ corresponding to the tMP_1. Keep taking the next frame in U_{1-q} and add it into tMP_1 until the error exceeds a fixing threshold.

Step 3 Rebuild DM_1, which includes A, B and F_0:

$$\begin{bmatrix} A \\ B \\ F_0 \end{bmatrix}_{3\times 57} \approx \begin{bmatrix} A_{1-q} \\ B_{1-q} \\ F_{0,1-q} \end{bmatrix}_{3\times q} \Lambda_{q\times 57} V_{57\times 57}{}^{\mathrm{T}}$$

where $\Lambda_{q\times 57}$ is composed of the first q rows in Λ, and then the frames corresponding to tMP_1 are chosen from M to form the motion unit MP_1.

Step 4 Use MD_1 and MP_1 to initialize MC_1.

Step 5 Take $T_{\min}$ frames from the left U_{1-q} and the procedure similar with Step 2 and 3, and extract motion unit MP_i and dynamic model MD_i.

Step 6 Calculate the distance between the dynamic model MD_i and the existing dynamics model extracted from the motion clusters. If the minimal value is below a threshold, the MP_i is added into this motion cluster; otherwise MD_i and MP_i are applied to initialize MC_i, and a new motion cluster is created.

Step 7 Repeat Step 5 and 6 until the motion M is finished.

Step 8 The algorithm is stopped, if the entire motion library is finished; otherwise another motion is decomposed by using SVD and it is skipped to Step 5.

The distance between two dynamic models is formalized to be the summation of the distance between Roots and the distance between gestures:

$$dis(MD_i, MD_k) = \beta dis\_t(MD_i, MD_k) + (1-\beta) dis\_p(MD_i, MD_k)$$

where $dis\_t(MD_i, MD_k)$ represents the distance between Roots, $dis\_p(MD_i, MD_k)$ is the distance between gestures, β is the weight and set to be 0.3. Since it is meaningless to compare two motion's initial positions and orientation, $dis\_t(MD_i, MD_k)$ is defined to be the minimum value of translation vector T and rotation vector R:

$$dis\_t(MD_i, MD_k) = \min_{R,T} \sum_{t=1}^{L} \Big[\alpha\big[(R \cdot MD_{pit} + T) - MD_{pkt}\big)^2 + (1-\alpha)(R \cdot MD_{oit} - MD_{okt})^2\Big] \Big/ L$$

where MD_{pit} is MD_i's position at time t, MD_{oit} is MD_i's orientation at time t. L is the time interval in calculating distance and it is set to be 6, α reflects the effects of the position parameter and orientation parameters in calculation procedure and it is set to be 0.8. $dis\_p(MD_i, MD_k)$ is the distance between the gestures contained in $\boldsymbol{MD}_i$ and $\boldsymbol{MD}_k$:

$$dis\_p(\boldsymbol{MD}_i, \boldsymbol{MD}_k) = \sum_{t=1}^{L} \sum_{j} w_j (\boldsymbol{MD}_{jit} - \boldsymbol{MD}_{jkt})^2 \Big/ L$$

where $\boldsymbol{MD}_{jit}$ is joint j's rotation in $\boldsymbol{MD}_i$ at time t, w_j reflects the joint j's effect.

7.2.1.2 Motion Graph Construction

Based on previous research [5,6], we adopt the first-order Markov chain to model the motion. Each state is formalized to be a motion cluster and the next state is only determined according to current state. But our approach is different with Lee, et al.'s work [6] which uses one frame to be a state. Our approach uses one motion cluster to be a motion state and it reduces the correspondence between current state and other states, so the first-order Markov chain is more adaptive in modeling motion. Transferring from state i to state j is formalized to be the value of connectivity between motion cluster $\boldsymbol{MC}_i$ and $\boldsymbol{MC}_j$. Each motion in the motion library is defined to be a first-order Markov chain, so the entire motion library can be represented by a motion graph $\boldsymbol{G}$:

$$\boldsymbol{G} = \{\boldsymbol{V}, \boldsymbol{E}\}$$

$$\boldsymbol{V} = \{\boldsymbol{MC}_1, \boldsymbol{MC}_2, \cdots, \boldsymbol{MC}_{N_{MC}}\}$$

$$\boldsymbol{E} = \{e_{ij} = P(\boldsymbol{MC}_j \mid \boldsymbol{MC}_i) \mid 1 \leqslant i, j \leqslant N_{MC}\}$$

In motion graph $\boldsymbol{G}$ each vertex is a motion cluster, the directed edge e_{ij} from vertex $\boldsymbol{MC}_i$ to $\boldsymbol{MC}_j$ is defined to be the value of connectivity. A usual approach used in calculating e_{ij} is statistic. For example, Li, et al.[5] counts how many times the connection between motion cluster $\boldsymbol{MC}_i$ and $\boldsymbol{MC}$ appears in the motion library and the result is used as the initial value for e_{ij}. The statistical method shows the dynamics features between original motion data. Its primary weakness is that the potential motion generation space in the constructed motion graph is limited by the original motion library. Another method in calculating connectivity [6,7] is to calculate the similarity between frames $\boldsymbol{F}_i$ and $\boldsymbol{F}_j$. Similarly we define e_{ij} to be the maximum similarity between motion units in $\boldsymbol{MC}_i$ and the motion units in $\boldsymbol{MC}_j$. This approach takes the actual connectivity between two motion clusters. In order to retain original motion's dynamic features and the size of the underlying space for motion synthesis, we adopt both these two statistical approaches to calculate e_{ij}:

$$e_{ij} = \alpha e_{ij}^{\mathrm{S}} + (1-\alpha) e_{ij}^{\mathrm{P}}$$

$$e_{ij}^{\mathrm{S}} = \sum_{k=1}^{N_{MP}-1} \delta(\boldsymbol{MP}_k \in \boldsymbol{MC}_i)\delta(\mathbf{MP}_{k+1} \in \boldsymbol{MC}_j)$$

$$e_{ij}^{\mathrm{P}} = \max_{\boldsymbol{MP}_{ik} \in \boldsymbol{MC}_i, \boldsymbol{MP}_{jl} \in \boldsymbol{MC}_j} sim(\boldsymbol{MP}_{ik}, pre(\boldsymbol{MP}_{jl}), L)$$

where e_{ij}^{S} represents the connecting times between the motion units in cluster $\boldsymbol{MC}_i$ and $\boldsymbol{MC}_j$ in the original motion library. $\delta(\boldsymbol{MP}_k \in \boldsymbol{MC}_i)$ is 1 if $\boldsymbol{MP}_k$ belongs to $\boldsymbol{MC}_i$, otherwise it is 1. N_{MP} is the number of motion units extracted from motion library. e_{ij}^{P} is the maximum similarity between the motion units in $\boldsymbol{MC}_i$ and $\boldsymbol{MC}_j$. In calculating the connectivity from one motion unit to another motion unit, the hinder part is more important than the frontal part, so only the hinder parts in both two motions are used in calculating similarity, which is defined below:

$$sim(\boldsymbol{M}_i, \boldsymbol{M}_k) = \beta\, sim\_t(\boldsymbol{M}_i, \boldsymbol{M}_k) + (1-\beta)\ sim\_p(\boldsymbol{M}_i, \boldsymbol{M}_k)$$

$$sim\_t(\boldsymbol{M}_i, \boldsymbol{M}_k) = \exp\left\{-\min_{\boldsymbol{R},\boldsymbol{T}} \sum_{t=1}^{L} \left[\alpha((\boldsymbol{RM}_{\mathrm{p}it} + \boldsymbol{T}) - \boldsymbol{M}_{\mathrm{p}kt})^2 + (1-\alpha)(\boldsymbol{RM}_{\mathrm{o}it} - \boldsymbol{M}_{\mathrm{o}kt})^2\right] \Big/ L\right\}$$

$$sim\_p(\boldsymbol{M}_i, \boldsymbol{M}_k) = \exp\left[-\sum_{t=1}^{L}\sum_{j} w_j(\boldsymbol{M}_{jit} - \boldsymbol{M}_{jkt})^2 \Big/ L\right]$$

where $sim(\boldsymbol{M}_i, \boldsymbol{M}_k)$ represents the similarity between $\boldsymbol{M}_i$ and $\boldsymbol{M}_k$, and $sim\_t(\boldsymbol{M}_i, \boldsymbol{M}_k)$ represents the similarity of Roots' motion. $\boldsymbol{M}_{\mathrm{p}it}$ and $\boldsymbol{M}_{\mathrm{o}it}$ represent Root's position and orientation in frame i. $sim\_t(\boldsymbol{M}_i, \boldsymbol{M}_k)$ defines the maximum similarity between the Roots in $\boldsymbol{M}_i$ and $\boldsymbol{M}_k$ when the rotation vector $\boldsymbol{R}$ and translation vector $\boldsymbol{T}$ are optimal [8]. $sim\_p(\boldsymbol{M}_i, \boldsymbol{M}_k)$ is the similarity between gesture $\boldsymbol{M}_i$ and $\boldsymbol{M}_k$. $\boldsymbol{M}_{jit}$ is the joint j's rotation vector at time t in $\boldsymbol{M}_i$, and w_j represents joint j's effect. At last the initialization result is united and $\sum_{j=1}^{N_{MC}} e_{ij} = 1$, in which N_{MC} is the number of vertexes in motion graph $\boldsymbol{G}$. Additionally, if the former motion unit is not found for $\boldsymbol{MP}_j$, the connectivity is set to be 0.5.

7.2.2 3D Motion Generation Based on Motion Graph

In this section the two-step motion synthesizing framework is described in detail. The new resolutions are proposed in two challenging domains by using this framework: (1) propose the random motion sampling techniques and create a series of similar motions for crowd animation; (2) propose motion path synthesizing techniques and create the motion, in which the character moves in a planed path.

7.2.2.1 Random Motion Sampling

Crowd simulation is always a challenging topic and can be used in many fields [9-13]. Here we concentrate on generating motion for crowd simulation. In the crowd, each character's motion should be similar with other characters but still has its own features. Using traditional key-frame techniques to create motion for each character is a tedious job. Although the MoCap data provide a feasible way in creating photorealistic animation, it is not appropriate for crowd simulation, since the generated motion will have uniform motion styles and it is impossible to capture the motion for every character. Here a new idea for creating crowd animation will be put forward: firstly an initial motion cluster or motion series are designated by the animator, and then a motion path composed of motion clusters is chosen in the motion graph, after that the random sampling techniques are adopted to create a series of motion units from the motion cluster path. These motion units are smoothed to form a new motion.

7.2.2.2 Motion Path Generation

If the animator only designates the initial motion cluster, the next frame in the motion path is chosen to be the cluster having the largest connectivity in the motion graph. This procedure repeats many times until the motion path $\boldsymbol{MS}_k = \{\boldsymbol{MC}_{k1}, \boldsymbol{MC}_{k2}, \cdots, \boldsymbol{MC}_{kN}\}$ meets the requested length.

If the animator designates a series of key motion clusters $\boldsymbol{MS}_s = \{\boldsymbol{MC}_{s1}, \boldsymbol{MC}_{s2}, \cdots, \boldsymbol{MC}_{sN}\}$, the motion path can be formed by synthesizing the interval motions. For example, the sub-path $\boldsymbol{MS}_{s1,s2}$ connecting key motion clusters $\boldsymbol{MC}_{s1}$ and $\boldsymbol{MC}_{s2}$ can be solved by such a optimization problem:

$$\begin{aligned}\boldsymbol{MS}_{s1,s2} = \arg\max_{\Pi} P(&\boldsymbol{MC}_{k1}, \boldsymbol{MC}_{k2}, \cdots, \boldsymbol{MC}_{kn} \mid \\ &\boldsymbol{MC}_{k1} = \boldsymbol{MC}_{s1}, \boldsymbol{MC}_{kn} = \boldsymbol{MC}_{s2}, \boldsymbol{G})\end{aligned}$$

where Π represents the gathering of all the paths from $\boldsymbol{MC}_{s1}$ to $\boldsymbol{MC}_{s2}$ in motion graph $\boldsymbol{G}$. Since the motion graph is based on first-order Markov chain [6], the optimization problem can be transformed to search the shortest path in a directed graph:

$$\begin{aligned}\boldsymbol{MS}_{s1,s2} &= \arg\max_{\Pi} P(\boldsymbol{MC}_{k1}, \boldsymbol{MC}_{k2}, \cdots, \boldsymbol{MC}_{kn} \mid \boldsymbol{MC}_{k1} = \boldsymbol{MC}_{s1}, \boldsymbol{MC}_{kn} = \boldsymbol{MC}_{s2}, \boldsymbol{G}) \\ &= \arg\max_{\Pi} P(\boldsymbol{MC}_{k2} \mid \boldsymbol{MC}_{s1}) P(\boldsymbol{MC}_{k3} \mid \boldsymbol{MC}_{k2}) \cdots P(\boldsymbol{MC}_{s2} \mid \boldsymbol{MC}_{kn-1}) \\ &= \arg\max_{\Pi} \lg P(\boldsymbol{MC}_{k2} \mid \boldsymbol{MC}_{s1}) P(\boldsymbol{MC}_{k3} \mid \boldsymbol{MC}_{k2}) \cdots P(\boldsymbol{MC}_{s2} \mid \boldsymbol{MC}_{kn-1}) \\ &= \arg\min_{\Pi} [-(\lg P(\boldsymbol{MC}_{k2} \mid \boldsymbol{MC}_{s1}) + \lg P(\boldsymbol{MC}_{k3} \mid \boldsymbol{MC}_{k2}) + \\ &\quad \cdots + \lg P(\boldsymbol{MC}_{s2} \mid \boldsymbol{MC}_{kn-1}))]\end{aligned}$$

where $P(\boldsymbol{MC}_{kj} \mid \boldsymbol{MC}_{ki})$ represents the connectivity from $\boldsymbol{MC}_{ki}$ to $\boldsymbol{MC}_{kj}$ in motion graph. A new directed graph is constructed to solve the above problem and it uses each motion cluster to be vertex and $-\lg P(\boldsymbol{MC}_{kj} \mid \boldsymbol{MC}_{ki})$ to be the edge weight from $\boldsymbol{MC}_{ki}$ to $\boldsymbol{MC}_{kj}$. Thus, the above optimization problem is transferred to be a shortest path searching problem. Dijkstra algorithm [14] can perfectly solve this problem and the time complexity is $O(N_{\mathrm{MC}}^2)$, in which N_{MC} is the number of motion graph $\boldsymbol{G}$'s vertexes.

After getting the motion path, the motion unit sequence is extracted.

There are more than one unit in each motion cluster, therefore the random sampling strategy is adopted to extract unit from the cluster. One motion path can generate a series of motion unit sequences, which have similar motions but different styles.

7.2.2.3 Motion Unit Sequence Connecting

After generating the sequence, each motion unit should be smoothed. Let $\boldsymbol{MR}$ be the motion unit sequence formalized as $\boldsymbol{MR}=\{\boldsymbol{MP}_{l1}, \boldsymbol{MP}_{l2}, \cdots, \boldsymbol{MP}_{lN}\}$; let $\boldsymbol{MP}_{1\text{-}i}$ be the motion by connecting motion sequences from $\boldsymbol{MP}_{l1}$ to $\boldsymbol{MP}_{li}$, $\boldsymbol{MP}_{l,i+1}$ is the next motion unit connecting to $\boldsymbol{MP}_{1\text{-}i}$, and the algorithm is as following.

Step 1 Translate $\boldsymbol{MP}_{l,i+1}$, let its first frame's Root position superpose the Root positon of $\boldsymbol{MP}_{1\text{-}i}$'s last frame .

Step 2 Rotate $\boldsymbol{MP}_{l,i+1}$, let its first frame's Root orientation be parallel with that of the last frame.

Step 3 Form a motion matrix $\boldsymbol{M}'_{2T_b\times 57}$ by $\boldsymbol{MP}_{1\text{-}i}$'s last T_b frames and $\boldsymbol{MP}_{l,i+1}$'s first T_b frames, and use the Gaussian convolution template to smooth the $\boldsymbol{M}'_{2T_b\times 57}$:

$$\boldsymbol{M}_{2T_b\times 57}=\boldsymbol{G}_{2T_b\times 2T_b}\boldsymbol{M}'_{2T_b\times 57}$$

$$\boldsymbol{G}_{2T_b\times 2T_b}=\begin{bmatrix} g_0(0) & g_0(1) & \cdots & g_0(2T_b-1) \\ g_1(-1) & g_1(0) & \cdots & g_1(2T_b-2) \\ \vdots & \vdots & & \vdots \\ g_{2T_b-1}(-2T_b+1) & g_{2T_b-1}(-2T_b+2) & \cdots & g_{2T_b-1}(0) \end{bmatrix}$$

$$g_i(t)=p(t)\Big/\sum_{t=-i}^{2T_b-1-i}p(t)$$

where $p(t)$ represents the standard Gaussian distribution.

Step 4 Use the motion transformation matrix $\boldsymbol{M}_{2T_b\times 57}$ to renew the motion $\boldsymbol{MP}_{1\text{-}i}$ and motion unit $\boldsymbol{MP}_{l,i+1}$. First use $\boldsymbol{M}_{2T_b\times 57}$'s first T_b frames to take the place of $\boldsymbol{MP}_{1\text{-}i}$'s last T_b frames, then use $\boldsymbol{M}_{2T_b\times 57}$'s last T_b frames to take the place of $\boldsymbol{MP}_{l,i+1}$'s first T_b frames.

Step 5 Connect $\boldsymbol{MP}_{l,i+1}$ to $\boldsymbol{MP}_{1\text{-}i}$ and create anew motion $\boldsymbol{MP}_{1\text{-}i+1}$.

7.2.2.4 Motion Path Synthesis

It is an important task to generate a path for the character in virtual reality, animation and 3D games. Since it is hard to guide the characters along different path, Gleicher[15] proposed a motion path editing method, which can help the animator to edit the path of existing motions interactively. However this method often makes the motion distortion, especially when huge differences exit between the new path and that of original motion. Therefore, based on the motion graph's synthesizing framework, we adopt a motion path synthesizing technique to create a motion with a new path.

Let $\boldsymbol{R}$ be the path and the motion along this path is created by using the following two steps: firstly find the motion unit sequence, which corresponds to path $\boldsymbol{R}$ in the motion graph, and then connect these motion units according to $\boldsymbol{R}$.

7.2.2.5 Motion Unit Sequence Extraction

We adopt the steps similar to Sect. 7. 2. 1. 1 to extract motion unit sequence: firstly a motion path is explored in the motion graph, and then a motion unit sequence is chosen from the motion clusters. Here the best motion path should satisfy the following two conditions: matching the motion path and the adjacent motion unit (cluster) should be connected naturally. According to these two conditions, the best motion path $\boldsymbol{MS}$ is:

$$\boldsymbol{MS}=\arg\max_{\Pi} P(\boldsymbol{MC}_{k1},\boldsymbol{MC}_{k2},\cdots,\boldsymbol{MC}_{kn}\mid \boldsymbol{MC}_{k1}=\boldsymbol{MC}_{s1},\boldsymbol{R},\boldsymbol{G})$$

Similarly, according to the motion graph based on first-order Markov chain, the above optimization problem is transferred to find the shortest path problem in a directed graph:

$$\begin{aligned}\boldsymbol{MS} &=\arg\max_{\Pi} P(\boldsymbol{MC}_{k1},\boldsymbol{MC}_{k2},\cdots,\boldsymbol{MC}_{kn}\mid \boldsymbol{MC}_{k1}=\boldsymbol{MC}_{s1},\boldsymbol{R},\boldsymbol{G})\\ &=\arg\max_{\Pi} P(\boldsymbol{MC}_{k2}\mid \boldsymbol{MC}_{s1})P(\boldsymbol{MC}_{k2}\mid \boldsymbol{R},\boldsymbol{S}_{k1})\cdots\\ &\quad P(\boldsymbol{MC}_{kn}\mid \boldsymbol{MC}_{k,n-1})P(\boldsymbol{MC}_{kn}\mid \boldsymbol{R},\boldsymbol{S}_{k,n-1})\\ &=\arg\max_{\Pi} \lg\,[P(\boldsymbol{MC}_{k2}\mid \boldsymbol{MC}_{s1})P(\boldsymbol{MC}_{k2}\mid \boldsymbol{R},\boldsymbol{S}_{k1})\cdots\\ &\quad P(\boldsymbol{MC}_{kn}\mid \boldsymbol{MC}_{k,n-1})P(\boldsymbol{MC}_{kn}\mid \boldsymbol{R},\boldsymbol{S}_{k,n-1})]\\ &=\arg\min_{\Pi}[-(\lg\,(P(\boldsymbol{MC}_{k2}\mid \boldsymbol{MC}_{s1})P(\boldsymbol{MC}_{k2}\mid \boldsymbol{R},\boldsymbol{S}_{k1}))+\cdots+\\ &\quad \lg\,(P(\boldsymbol{MC}_{kn}\mid \boldsymbol{MC}_{k,n-1})P(\boldsymbol{MC}_{kn}\mid \boldsymbol{R},\boldsymbol{S}_{k,n-1})))]\end{aligned}$$

where $\boldsymbol{MC}_{s1}$ is the designated initial motion cluster, $\boldsymbol{G}$ is the motion graph, $P(\boldsymbol{MC}_{kj}\mid \boldsymbol{MC}_{ki})$ represents the connectivity from $\boldsymbol{MC}_{ki}$ to $\boldsymbol{MC}_{kj}$, $P(\boldsymbol{MC}_{kj}\mid \boldsymbol{R},\boldsymbol{S}_{ki})$ represents the matching between $\boldsymbol{MC}_{kj}$ and $\boldsymbol{S}_{ki}$. The above problem cannot be transferred to find the shortest problem in a directed graph, which is different from Sect. 7. 2. 1. 1, because the new constructed directed graph has infinite vertexes and no explicit end point.

The constructed directed graph is shown in Fig. 7. 14, where vertex $\boldsymbol{V}_0$ is the initial motion cluster, and other vertexes are defined as:

$$\boldsymbol{V}_{ij}=\boldsymbol{MC}_j \quad (1\leqslant j\leqslant N_{\mathrm{MC}})$$

where N_{MC} is the gross number of motion cluster, and $\boldsymbol{V}_{ij}$ is the jth vertex in the ith layer. The edge from vertex $\boldsymbol{V}_{ij}$ to vertex $\boldsymbol{V}_{i+1,k}$'s weight is $-\lg\,(P(\boldsymbol{MC}_{ki}\mid \boldsymbol{MC}_{k,i-1})P(\boldsymbol{MC}_{k,i}\mid \boldsymbol{R},\boldsymbol{S}_{k,i-1}))$, in which $\boldsymbol{MC}_{ki}$ and $\boldsymbol{MC}_{k,i-1}$ are

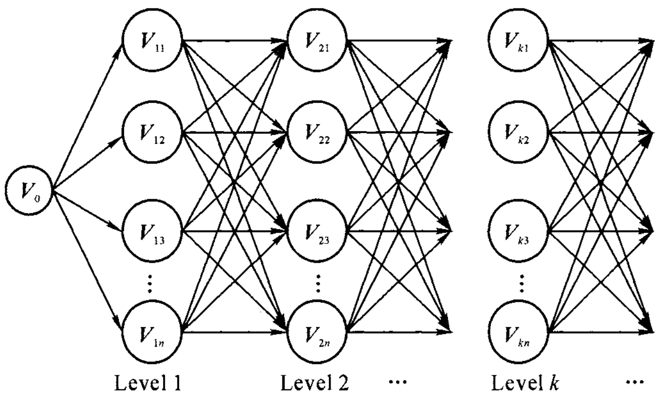


Fig. 7. 14 Directed graph with layers

the corresponding motion clusters to vertex $\boldsymbol{V}_{i+1,k}$ and $\boldsymbol{V}_{ij}$. $\boldsymbol{S}_{k,i-1}$ follows the path from vertex $\boldsymbol{V}_0$ to $\boldsymbol{V}_{ij}$ in path $\boldsymbol{R}$. Since different path will affect the value of $\boldsymbol{S}_{k,i-1}$, the weight of the edge connecting two vertexes is changing dynamically. An extended Dijkstra algorithm is designed by us to solve the above optimization problem.

Let $\boldsymbol{S}$ be the set, which collects all the vertexes in the shortest path from vertex $\boldsymbol{V}_0$. Let $\boldsymbol{Q}$ be the set, which collects all the vertexes in the current shortest path. The algorithm is as follows.

```
(1) Initialization
    $V_0.d=0$
        // set the weight of the shortest path from $V_0$ to this vertex
    $V_0.mp=Get\text{-}Fittest\text{-}MP(V_0, R, P_0)$
        //find the representative motion primitive of $V_0$ that is
        //fittest to the path segment of $R$ starting at $P_0$
        //$P_0$ is the start location of $R$
    $V_0.p\_end=Reach\text{-}end(V_0, R, P_0)$
        // calculate the end location after the representative
        // motion primitive in $V_0$ has been adapted to $R$
    $Q \leftarrow V_0$
        //add $V_0$ to $Q$
(2) While the end location of $R$ has not been approximated enough
    $V_c= Extract\text{-}Min(Q)$
        //fetch a vertex with the minimal best estimation in $Q$ as the current vertex
    $S \leftarrow V_c$
        //add $V_c$ to $S$
    $Remove(V_c, Q)$
        // remove $V_c$ from $Q$
    for each vertex $V_i$ in the next level to $V_c$
      $V_i.pre=V_c$
          // set $V_c$ as the precedent of $V_i$
    $V_i.d = V_c.d + (-\lg(P(V_i|V_c)P(V_i|R, V_c.p\_end))$
        // calculate the current shortest path weights from $V_0$ to $V_i$
    $V_i.mp = Get\text{-}Fittest\text{-}MP(V_i, R, V_c.p\_end)$
        // find the representative motion primitive in $V_i$
        // that is fittest to the path segment of $R$ starting at $V_c.p\_end$
    $V_i.p\_end = Reach\text{-}end(V_i, R, V_c.p\_end)$
        //calculate the end location after the shortest
        //path from $V_0$ to $V_i$ has been adapted to $R$
    if $V_i \in Q$,
      Update the $V_i$ in $Q$
    else
        $Q \leftarrow V_i$
      endif
    endfor
  endwhile
```

$P(V_i|R, V_c.p\_end)$ represents the matching between V_i and the path in R from $V_c.p\_end$:

$$P(V_i|R, V_c.p\_end) = \max_{MP_{ij} \in V_i} P(MP_{ij}|R, V_c.p\_end)$$

We adopt the minimum changing strategy to calculate $P(MP_{ij}|R, V_c.p\_end)$, which is defined to coordinate the motion unit MP_{ij}. The specific coordinating algorithm is shown in the next section. *Reach-end*$(V_i, R, V_c.p\_\text{end})$ calculates the motion unit sequence from V_0 to V_i and the specific algorithm is shown in the next section. *Get-Fittest-MP*$(V_i, R, V_c.p\_end)$ finds the optimal motion unit in vertex V_i:

$$Get\text{-}Fittest\text{-}MP(V_i, R, V_c.p\_end) = \arg\max_{MP_{ij} \in V_i} P(MP_{ij}|R, V_c.p\_end)$$

After the above algorithm finishes, the motion path can be explored by using reversing deduction: from the last vertex, join S to the initial vertex V_0, and transpose the extracted motion cluster sequence to get the motion path. The optimal motion unit in each motion cluster is chosen to form the motion unit sequence.

7.2.2.6 Connecting of Motion Unit Sequence and Path Fitting

After getting the required motion unit sequence, these motion units are connected according to the path, so the required motion can be generated. Let M be the motion, which is formed by the motion units before MP_{i+1} according to path R, PS is M's end point. The algorithm is as follows.

Step 1 The first frame in MP_{i+1} is chosen to be extended.

Step 2 Translate the current frame and let its Root be PS.

Step 3 Calculate path R in PS's tangent, and rotate current frame to be parallel with the tangent. If the current frame is the first frame in

$\boldsymbol{MP}_{i+1}$, it replaces the end frame in $\boldsymbol{M}$, otherwise the frame is added into $\boldsymbol{M}$ to be the end frame.

Step 4 If all the frames in $\boldsymbol{MP}_{i+1}$ are managed completely, then it is the end; otherwise $\boldsymbol{PS}$ are moved following $\boldsymbol{R}$, and the moving distance is the distance between the current frame and the last frame. The next frame will become the current frame and go to Step 2.

In order to get the motion with natural transition, Gaussian convolution template is adopted to refine the results.

$$\begin{bmatrix} \boldsymbol{M}(m-T_b) \\ \boldsymbol{M}(m-T_b+1) \\ \vdots \\ \boldsymbol{M}(m+T_b) \end{bmatrix} = \boldsymbol{G}_{(2T_b+1)\times(2T_b+1)} \begin{bmatrix} \boldsymbol{M}(m-T_b) \\ \boldsymbol{M}(m-T_b+1) \\ \vdots \\ \boldsymbol{M}(m+T_b) \end{bmatrix}$$

$$\boldsymbol{G}_{(2T_b+1)\times(2T_b+1)} = \begin{bmatrix} g_0(0) & g_0(1) & \cdots & g_0(2T_b) \\ g_1(-1) & g_1(0) & \cdots & g_1(2T_b-1) \\ \vdots & \vdots & & \vdots \\ g_{2T_b-1}(-2T_b) & g_{2T_b-1}(-2T_b+1) & \cdots & g_{2T_b-1}(0) \end{bmatrix}$$

$$g_i(t) = p(t) \Big/ \sum_{t=-i}^{2T_b-i} p(t)$$

where m is the frames connecting $\boldsymbol{M}$ and $\boldsymbol{MP}_{i+1}$, $[m-T_b, m+T_b]$ is the managing range in the new motion, and $p(t)$ represents the standard Gaussian distribution.

7.2.3 Results

The motion library, including 270 frames of normal walk and 246 frames of Catwalk, is used to test the motion synthesizing algorithm introduced above. Firstly 8 motion clusters are chosen by using greedy algorithm and each motion cluster is shown in Table 7.1. The connectivity between motion clusters is shown in Table 7.2. According to these data, a motion graph containing 8 vertexes can be constructed and each vertex represents a motion cluster. The weight in each edge e_{ij} represents the connectivity from cluster $\boldsymbol{MC}_i$ to $\boldsymbol{MC}_j$.

Table 7.1 Motion cluster

Motion cluster ID	Motion primitive ID	Motion cluster ID	Motion primitive ID
$\boldsymbol{MC}_1$	1,5,9	$\boldsymbol{MC}_5$	13,17,21
$\boldsymbol{MC}_2$	2,6,10	$\boldsymbol{MC}_6$	14,18,22
$\boldsymbol{MC}_3$	3,7,11	$\boldsymbol{MC}_7$	15,19,23
$\boldsymbol{MC}_4$	4,8,12	$\boldsymbol{MC}_8$	16,20

Table 7.2 Connectivity between motion clusters

	MC_1	MC_2	MC_3	MC_4	MC_5	MC_6	MC_7	MC_8
MC_1	0.15926	0.34525	0.15608	0.13728	0.11148	0.05440	0.03569	0.00054
MC_2	0.07274	0.19349	0.41504	0.11105	0.04602	0.00063	0.12664	0.03436
MC_3	0.17830	0.15790	0.11202	0.34285	0.10163	0.00359	0.00048	0.10324
MC_4	0.31031	0.18492	0.11327	0.18556	0.13796	0.02318	0.00037	0.04444
MC_5	0.00048	0.12003	0.06427	0.03981	0.16930	0.37751	0.16918	0.05943
MC_6	0.00037	0.10271	0.14585	0.05132	0.11714	0.15330	0.31367	0.11566
MC_7	0.00062	0.05757	0.08013	0.14393	0.10421	0.04082	0.11969	0.45303
MC_8	0.09650	0.13167	0.00087	0.07141	0.50771	0.12940	0.04383	0.01861

7.2.3.1 Random Motion Sampling

Two key standards for crowd simulation are reality and characteristic. Each character's motion must be natural and all the characters' motions should be similar while having their own characteristics. In the motion synthesizing framework introduced above, the animator only needs to designate an initial motion cluster and the crowd motion can be generated: the system explores an optimal motion path from the motion graph according to the designated initial motion cluster; then the random motion sampling strategy is adopted to choose a series of motion units from the motion path; at last the motion units are smoothed to be the requested motion. For example, MC_1 is designated to be the initial motion cluster and the system can extract the motion path $\{MC_1, MC_2, MC_3, MC_4, MC_1, MC_2, MC_3, MC_4, MC_1, MC_2\}$ from motion graph. A series of motion can be created by using this motion path, as shown in Fig. 7.15.

The motion units in motion cluster have similar dynamic characteristics, however every motion has different unit sequence, and the motion is constructed from the MoCap data. The weakness is that the motion path always falls into a small range in the motion graph. In the above example, the motion path is constructed from $\{MC_1, MC_2, MC_3, MC_4\}$ repeatedly.

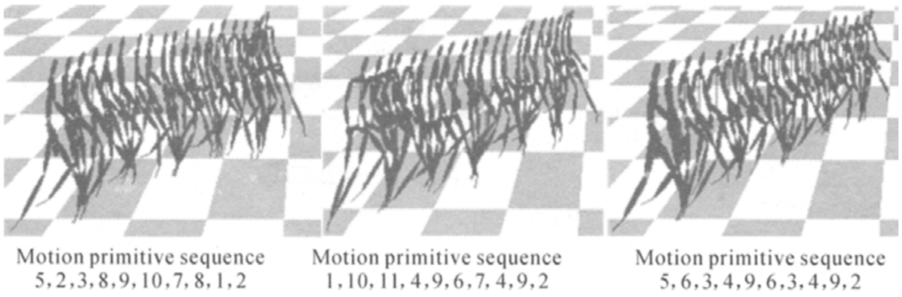

Motion primitive sequence
5,2,3,8,9,10,7,8,1,2
Motion primitive sequence
1,10,11,4,9,6,7,4,9,2
Motion primitive sequence
5,6,3,4,9,6,3,4,9,2

Fig. 7.15 Random motion sampling

This may reduce the underlying motion synthesis space. There are two reasons: the fixing period exists in the original motion, and the limitation exists in the connectivity calculation method. For example, a walk motion contains four periods and each period has three different motion units. According to the motion unit extraction algorithm, three motion clusters can be extracted and each cluster contains four motion units as shown in Fig. 7. 16. By using the connectivity calculating algorithm illustrated in Sect. 7. 2. 1, it is possible that e_{12} is larger than other edges from MC_1; e_{23} is larger than other edges from MC_2; e_{31} is larger than other edges from MC_3. Thus, if the three motion clusters shown in Fig. 7. 16 are chosen into the motion path, this path will fall into a dead loops {MC_1, MC_2, MC_3}.

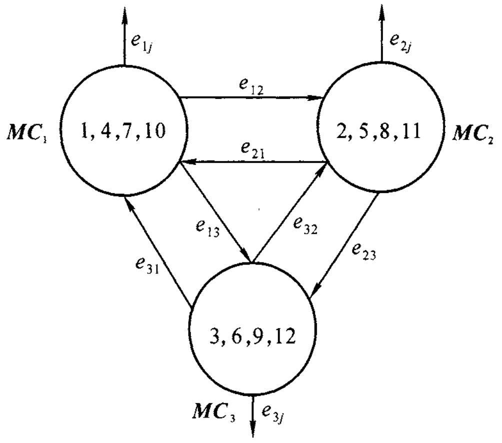


Fig. 7. 16 Extracting motion clusters from motion period

A feasible solving method is to utilize a noise generator to motivate the motion path from the dead loop. This method will damage motion's reality, since adding noises may induce the randomly choosing of motion cluster. An interactive approach is adopted to solve this problem: firstly the animator inserts a key motion cluster sequence in the initial motion path, and then the system adopts the algorithm elaborated in Sect. 7. 2. 1. 1 to generate the motion cluster sequence between two neighboring ***MC***. Thus, the dead loop can be broken and the motion reality can be guaranteed. For example, {MC_8, MC_3} is inserted behind the original ***MC*** sequences {MC_1, MC_2, MC_3, MC_4, MC_1}, and {MC_1, MC_8}, {MC_8, MC_3}, {MC_3, MC_2} are extended to be {MC_1, MC_2, MC_7, MC_8}, {MC_8, MC_2, MC_3} and {MC_3, MC_2}, so the original ***MC*** sequence turns into {MC_1, MC_2, MC_3, MC_4, MC_1, MC_2, MC_7, MC_8, MC_2, MC_3, MC_2, MC_3, MC_4, MC_1, MC_2}.

Obviously the new ***MC*** sequence is much longer than the original one and the animator can choose one segment to be the result. For example, ten motion clusters are chosen from this ***MC*** sequence, and a series of new motions can be created. Fig. 7. 17 shows one of them.

7.2.3.2 Motion Path Synthesizing

According to the framework given above, we propose a motion path synthesizing algorithm to generate the motion along the programmed path.

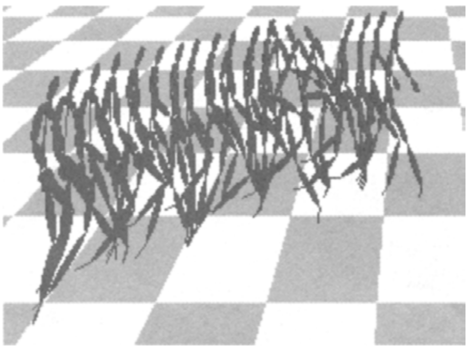


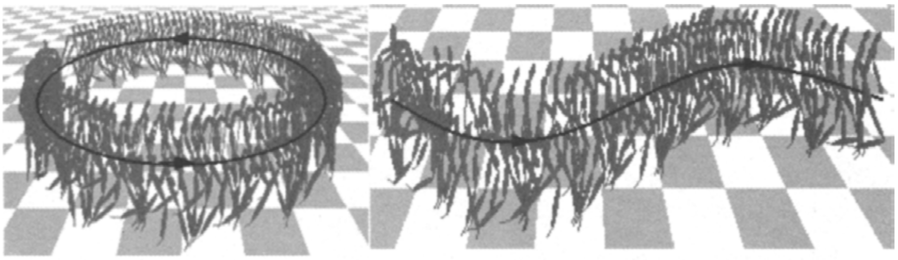
Fig. 7.17 The generated motion after refining ***MC*** sequence, and the corresponding motion unit sequence is {1, 2, 7, 8, 5, 6, 19, 20, 2, 3}

This algorithm has two steps: first a motion unit sequence is extracted from the motion graph, then the units are connected following the path.

For example, when the user designates a round path, the system extracts a motion sequence {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 2, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8, 9,10} from the motion graph. A motion having 915 frames is created, as shown in Fig. 7. 18(a). Similarly, a motion having 447 frames can be created and it moves along sine curve shown in Fig. 7. 18 (b). The units forming this motion are {13, 14, 15, 16, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 9, 10, 11, 12, 9, 10, 11, 12}.

From the above two examples, the new generated motion has more frames than the original motion library. Some motion units are used frequently in constructing new motion, so two motion units, which are not close, must have smooth transition. The motion synthesizing framework adopts two strategies to realize the transition: first, the motion unit sequence whose smooth transition is possibly realized is chosen; second, adopt Gaussian convolution template to manage two neighboring motion units. The experimental results show the effect of these strategies.

This chapter introduces a motion synthesis framework and two techniques: random motion sampling and motion path synthesis. The main



(a) Walking along roundness (b) Walking along sine curve

Fig. 7.18 Motion path synthesizing

contribution is to propose a novel motion graph concept, which is different from the previous work. It adopts motion clusters to be the vertexes but not single frame or the entire motion. By using this approach, the motion's reality can be retained and the underlying motion space can be effectively enlarged. The characteristic of the new motion graph is to retain the original motion in the cluster. Another contribution is to propose new techniques in two challenging areas: one is the random motion sampling technique in crowd animation; the other is motion path synthesis technique along the programmed path.

7.3 Automatic Synthesis and Editing of Motion Styles

Motion synthesis and editing methods are extensively studied in recent years and many wonderful results have been achieved, but the techniques for synthesis and editing of motion styles have been rarely explored. In this chapter we propose a framework for automatic, real-time and quantitative synthesis and editing of human motion styles. In this framework, Principal Component Analysis (PCA) theory is used to map original styled human motions into subspaces, which can reduce computational complexity while reserving the intrinsic properties of original data. Synthesis and editing methods are applied in such subspaces and then motions with new styles can be reconstructed. As realistic human motions may have multiple styles, we also present a novel method to synthesize and edit motions with multiple styles.

In computer animation research, synthesis and editing methods for motion styles are very important: (1) in a large virtual environment, large numbers of characters with different motion styles are simultaneously needed; (2) in animation production systems, the animator can only get motions with desired styles by synthesizing or editing existing motion clips in MoCap database instead of capturing every motion directly; (3) for a motion with an exaggerated style which cannot be performed by the actor, synthesis and editing methods for motion styles are also needed.

But all of these are great challenges because motion data are tremendous and have high-dimensionality. For example, a motion with 30 frames and 51 DOFs is represented as a vector with 1,530 dimensions, semantics of motion data is not obvious. Relationship between original motion data and high-level semantics is complex. Motion styles are very abstract and have higher-level semantics, so it's even more difficult to uncover the relationship between them.

In order to overcome the computational complexity, data reduction method should be applied. Despite the data reduction, we reserved the intrinsic properties of original data as much as possible. PCA [16] is such a method. With PCA, a subspace can be constructed, which can be used to describe original data space approximately. In general, the dimension of PCA subspace is much smaller than that of original data space.

Motion style is an abstract concept, so quantitative analysis of motion style is different from that of low-level motion data. For example, while a walking motion with a velocity of 6 km/h has an explicit physical definition, the definition of a walking motion with "happy" or "sad" style is indistinct and subjective. Motion style is a subjective perception in human's mind, and the quantitative analysis of motion style is to make the processing more precise and the concept of styles more intuitive. For example, later we'll use a number between 0. 0 to 2. 0 to describe the "catty" degree of a walking motion. We do not relate this number to any physical definitions; its meaning lies in that we ensure that the average value of this number for "catty" training samples is 1. 0 and that for "natural" training samples is 0. 0, and that in the same context, motion with larger "catty" value has more obvious catty style. For a motion with "catty" value of 0. 3, it should lie between "natural" and "catty" samples and nearer to "natural" in catty degree. Motion styles only have meanings in comparison.

We aim to propose a framework for synthesis and editing of human motion styles, by which users can automatically and quantitatively synthesize and edit motions with different styles in real-time.

7. 3. 1 Motion Data Preprocessing

Human body is modeled by an articulated skeleton with some rigid limbs, which is the same as the human model used in Sect. 7. 1. 1. The walking motions with different styles are used to illustrate our idea, including catty walk, large-arm-swing walk, rocky walk, high-step walk, stomp walk and titanic walk. Also a number of "natural" walking motions are needed as references, which have moderate step length, height, and normal arm swing, but no motion style component mentioned above. In our experiments, an optical MoCap system is used to acquire human motion data.

We divide these walking motions into 6 groups, group 1 is "natural" walk and groups 2 to 6 correspond to catty walk, large-arm-swing walk, rocky walk. Two steps are adopted to preprocess the motion data.

Step 1 Extract a complete motion cycle from every motion clip and employ a time-warping method to scale the cycles to the same frame number.

Step 2 Filter out the translation and rotation of the root joint, which present the overall position of skeleton and have no relations with motion style.

After preprocessing, all training motion samples have the same frame number and DOFs correspond to a group of vectors in the same original data space. For testing motion samples, the same preprocessing is applied.

7.3.2 Motion Synthesis and Editing Algorithm for Single Style Component

7.3.2.1 PCA Subspace and Motion Style Vector

The style component "catty" is used for illustration. First we construct a PCA space based on a group of natural walk and a group of catty walk, and compute the points of these samples in the subspace. Let $\boldsymbol{a}_{\mathrm{n}}$ and $\boldsymbol{a}_{\mathrm{c}}$ be the center of the natural samples and catty samples respectively, then the vector $\boldsymbol{a}_{\mathrm{c}} - \boldsymbol{a}_{\mathrm{n}}$ is the direction that controls or affects motions' catty degree. Along the positive direction of this vector, the catty degree increases, and vice versa. In this way, we essentially find a relation between the motion data and catty degree, which is semantic. This is the base of the following algorithms.

Below we call the vectors, such as $\boldsymbol{a}_{\mathrm{c}} - \boldsymbol{a}_{\mathrm{n}}$, style (controlling) vectors in the PCA space.

7.3.2.2 Motion Synthesis Algorithm for Single Style Component

This algorithm is for automatically synthesizing a walking motion given a certain catty degree. First we quantify the catty degree. The mean of catty degrees of the catty training samples is 1.0, and the same quantum of the natural samples is 0.0. Some extent of extrapolation is allowed (see Fig. 7.19(a)).

The following is a formal description of the algorithm. In this description, c is the expected catty degree of the synthesized motion, S is the PCA space, $\boldsymbol{a}_{\mathrm{c}}$ is the center of catty samples in the PCA space, and $\boldsymbol{a}_{\mathrm{n}}$ is the center of natural samples in the PCA space.

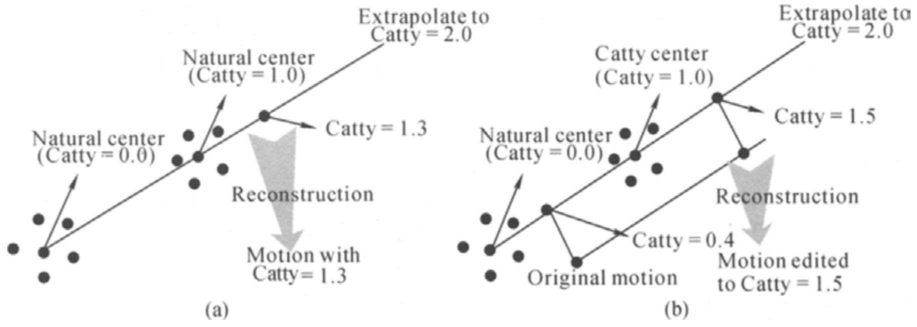

Extrapolate to Catty = 2.0
Natural center (Catty = 1.0)
Natural center (Catty = 0.0)
Catty = 1.3
Reconstruction
Motion with Catty = 1.3
(a)
Extrapolate to Catty = 2.0
Catty center (Catty = 1.0)
Natural center (Catty = 0.0)
Catty = 1.5
Reconstruction
Catty = 0.4
Original motion
Motion edited to Catty = 1.5
(b)

Fig. 7.19 (a) Motion synthesis algorithm for single style component. This figure illustrates the algorithm when the expected catty degree is 1.3; (b) Motion editing algorithm for single style component. This figure illustrates the case when the expected catty degree is 1.5

Algorithm 1 Motion Synthesis Algorithm for Single Style Component

```
begin initialize $c$, $S$, $\boldsymbol{a}_n$, $\boldsymbol{a}_c$
    $\boldsymbol{a} \leftarrow \boldsymbol{a}_n + c \times (\boldsymbol{a}_c - \boldsymbol{a}_n)$
    reconstruct the motion from $\boldsymbol{a}$ and return it
end
```

7.3.2.3 Motion Editing Algorithm for Single Style Component

This algorithm is for automatically editing an existing walking motion to a certain catty degree. The quantification and assumptions are the same as Algorithm 1. The following is a formal description, in which θ is an existing walking cycle, c is the expected catty degree of the edited motion, S is the PCA space, $\boldsymbol{a}_c$ is the center of catty samples in the PCA space, and $\boldsymbol{a}_n$ is the center of natural samples in the PCA space (see Fig. 7.19(b)).

Algorithm 2 Motion Editing Algorithm for Single Style Component

```
begin initialize $\theta$, $c$, $S$, $\boldsymbol{a}_n$, $\boldsymbol{a}_c$
  project $\theta$ into $S$ and get its coordinate point in PCA space as $\boldsymbol{a}_0$
  $c_0 \leftarrow (\boldsymbol{a}_0 - \boldsymbol{a}_n) \cdot (\boldsymbol{a}_c - \boldsymbol{a}_n) / \| \boldsymbol{a}_c - \boldsymbol{a}_n \|^2$
  $\boldsymbol{a} \leftarrow \boldsymbol{a}_n + (c - c_0) \times (\boldsymbol{a}_c - \boldsymbol{a}_n)$
  reconstruct the motion from $\boldsymbol{a}$ and return it
end
```

7.3.3 Motion Synthesis and Editing Algorithm for Multiple Style Components

Here we present a novel approach to synthesize and edit motions based on multiple styles. We take the style with two components "catty" and "large-arm-swing" as example. Consider a motion style description vector $\boldsymbol{MS} = [\textit{catty}, \textit{large-arm-swing}]$, in which the two components represent the two style aspects. We develop algorithms based on this style description vector.

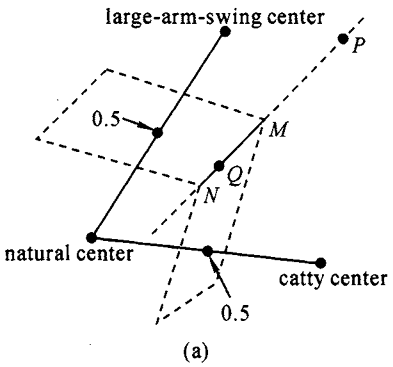
Things would be easier if different style components were orthogonal. In that case, all we need to do is happily divide an n-dimensional style problem into n single style problems and Algorithms 1 and 2 will do all the rest. In practice, this premise is rarely true. For example, if we synthesize a walking motion with catty degree 1.0 by Algorithm 1 and then edit this motion by Algorithm 2 to large-arm-swing degree 1.0, we won't probably get a final motion with $\boldsymbol{MS} = [1.0, 1.0]$, because the modification made by Algorithm 2 aiming at large-arm-swing style component also has effects on catty component. Therefore, we should consider different style components associatively rather than separately.

In the algorithms presented below, we assume that a PCA space S has been constructed based on all the training samples of natural, catty and large-arm-swing, and that $\boldsymbol{a}_n$, $\boldsymbol{a}_c$ and $\boldsymbol{a}_l$ are the center of natural, catty and large-arm-swing samples in the PCA space, respectively.

7.3.3.1 Motion Synthesis Algorithm for Multiple Style Components

This algorithm is for automatically synthesizing a walking motion given a style description vector $\boldsymbol{MS} = [c, l]$. Formally, we want to find such a point in PCA space that when projected to $\boldsymbol{a}_c - \boldsymbol{a}_n$, it has a catty degree c and when projected to $\boldsymbol{a}_l - \boldsymbol{a}_n$, it has a large-arm-swing degree l. Direct computation is possible. We first find two vectors $\boldsymbol{a}_c - \boldsymbol{a}_n$ and $\boldsymbol{a}_l - \boldsymbol{a}_n$, and then find the two hyper planes perpendicular to these two vectors, at last the intersection of these two hyper planes are the points that satisfy the formal requirement. There are two problems with this direct computation. First, in a space with dimension larger than 2, the intersection of the two hyper planes, if exists, consists of indefinite number of points (see in Fig. 7.20(a)), and even worse, we can hardly expect that a point far away from all the sample centers (such as P), after re-constructed to high-dimensional motions, still represents a valid walking motion. In experiments we did prove that these faraway points, when restored to motions, are nothing but messes. Our algorithm should be able to overcome this problem and then find one point in the space that not only satisfies the mathematical requirements but is also a representative walking motion, or rather, not too far from the training samples' centers (such as Q in Fig. 7.20 (a)). The second problem with the direct computation is that usually the computational complexity will be enlarged remarkably when the space's dimension increases.

Now we present our algorithms and show how our algorithms address the above two problems.



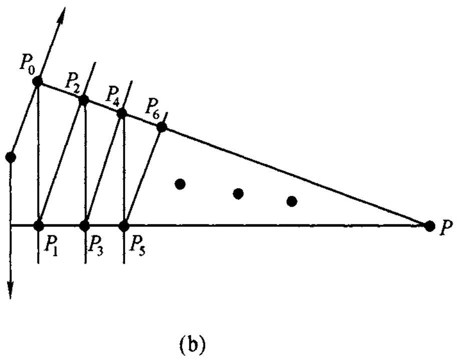


Fig. 7.20 (a) The non-singularity of the problem when the PCA subspace is three-dimensional and when the expected style description vector is [0.5, 0.5]; (b) The iterations when the angle between two style controlling vectors is close to 180°

Algorithm 3 Motion Synthesis Algorithm for Multiple Styles

```
begin initialize $c$, $l$, $S$, $\boldsymbol{a}_n$, $\boldsymbol{a}_c$, $\boldsymbol{a}_l$, threshold $\varepsilon$ and $\varepsilon'$
  $\beta \leftarrow$ the angle between $\boldsymbol{a}_c - \boldsymbol{a}_n$ and $\boldsymbol{a}_l - \boldsymbol{a}_n$
  if $abs(\beta - 180°) < \varepsilon'$
    print ("No solutions can be found.")
      return;
    end
  $\boldsymbol{a} \leftarrow c \times (\boldsymbol{a}_c - \boldsymbol{a}_n)$
  while (1)
    $c' \leftarrow (\boldsymbol{a} - \boldsymbol{a}_n) \cdot (\boldsymbol{a}_c - \boldsymbol{a}_n) / \| \boldsymbol{a}_c - \boldsymbol{a}_n \|^2$
    $l' \leftarrow (\boldsymbol{a} - \boldsymbol{a}_n) \cdot (\boldsymbol{a}_l - \boldsymbol{a}_n) / \| \boldsymbol{a}_l - \boldsymbol{a}_n \|^2$
    if $abs(c - c') < \varepsilon$ and $abs(l - l') < \varepsilon$
      break;
    end;
    if $(c - c') > \varepsilon$
      $\boldsymbol{a} \leftarrow \boldsymbol{a} + (c - c') \times (\boldsymbol{a}_c - \boldsymbol{a}_n)$
      continue;
    end
    if $(l - l') > \varepsilon$
      $\boldsymbol{a} \leftarrow \boldsymbol{a} + (l - l') \times (\boldsymbol{a}_l - \boldsymbol{a}_n)$
    end
  end
  reconstruct the motion from $\boldsymbol{a}$ and return it
end
```

Our algorithm takes an asymptotical approach. We want the algorithm to be well converging. Here a problem must be addressed. Mathematically, unless the two vectors $\boldsymbol{a}_c - \boldsymbol{a}_n$ and $\boldsymbol{a}_l - \boldsymbol{a}_n$ are exactly parallel, the algorithm will be converging. Here we consider the situation where the two vectors are not strictly parallel but have an angle close to 180°. In that case, the algorithm will stop, but only after many iterations, and the worst thing is that the motion returned will be very far from all the three groups of training samples, yielding a disordered motion (Fig. 7.20(b) is an example). In Fig. 7.20(b), the two arrows are the two style controlling vectors in subspace. Point P is the point that meets the formal requirements. Asymptotically, the algorithm finds $P_0, P_1, \cdots$ and finally it returns a point near P. But P is so faraway from all the samples that probably it only represents a mess instead of a walking motion. To overcome this problem, we simply add a decision at the beginning of our algorithm. If the angle between the two vectors is close to 180°, we say that no appropriate motion can be synthesized to satisfy the requirement. This is reasonable, since if the two vectors are nearly opposite, we can infer that the two corresponding style components have obvious opposite meaning. For example, for a walking motion, "catty" implies smaller transverse step width and larger lengthways step length, while "rocky" implies larger step width and smaller step length. These two style components have obvious opposite meanings so the style vectors in the PCA space corresponding to them will probably be opposite.

If the two style vectors in the space are not nearly opposite, our algorithm will stop after only a few iterations and return a motion. Note that since the algorithm initializes the motion to be a point along the first style controlling vector that is not far from the samples and the algorithm stops after only several iterations, the returned motion is not very far from the samples and therefore disorder is avoided.

7.3.3.2 Motion Editing for Multiple Style Component Algorithm

The editing algorithm for multiple styles is very similar to the synthesis algorithm. The only difference lies in the initialization. The following is a formal description, in which θ is the existing motion that we want to edit.

Algorithm 4 Motion Editing Algorithm for Multiple Styles

```
begin initialize $\theta$, $c$, $l$, $S$, $\boldsymbol{a}_n$, $\boldsymbol{a}_c$, $\boldsymbol{a}_l$, threshold $\varepsilon$ and $\varepsilon'$
  $\beta \leftarrow$ the angle between $\boldsymbol{a}_c - \boldsymbol{a}_n$ and $\boldsymbol{a}_l - \boldsymbol{a}_n$
  if $abs(\beta - 180°) < \varepsilon'$
    print ("No solutions can be found. ")
  return;
  end
project $\theta$ into $S$ and get its coordinate point in PCA space as $\boldsymbol{a}$
    while (1)
  $c' \leftarrow (\boldsymbol{a} - \boldsymbol{a}_n) \cdot (\boldsymbol{a}_c - \boldsymbol{a}_n) / \| \boldsymbol{a}_c - \boldsymbol{a}_n \|^2$
  $l' \leftarrow (\boldsymbol{a} - \boldsymbol{a}_n) \cdot (\boldsymbol{a}_l - \boldsymbol{a}_n) / \| \boldsymbol{a}_l - \boldsymbol{a}_n \|^2$
  if $abs(c - c') < \varepsilon$ and $abs(l - l') < \varepsilon$
    break;
  end;
  if $(c - c') > \varepsilon$
    $\boldsymbol{a} \leftarrow \boldsymbol{a} + (c - c') \times (\boldsymbol{a}_c - \boldsymbol{a}_n)$
    continue;
  end
  if $(l - l') > \varepsilon$
    $\boldsymbol{a} \leftarrow \boldsymbol{a} + (l - l') \times (\boldsymbol{a}_l - \boldsymbol{a}_n)$
  end
end
reconstruct the motion from $\boldsymbol{a}$ and return it
end
```

7.3.4 Results and Discussions

In order to show the power of our algorithm, we now present some experiments. The training samples are derived as follows. For each style component, we acquire 5 samples, so totally we capture 35 motion clips (30 clips for the six style components and 5 clips for the "natural walk"). All clips are acted by the same subject. Data preprocessing is conducted as described in Sect. 7. 3. 1, in which all clips are time-warped to 30 frames. The skeleton we use has 51 DOFs including the root joint. Therefore, after preprocessing, each training clip is a 1,350-dimensional vector in the original data space. All the experiments are implemented in Matlab on a PC with a PentiumIV 2. 4 GHz CPU and 512 MB RAM.

When we get results from algorithms, inverse kinematics methods are applied to eliminate the artifacts such as feet sliding [17].

7.3.4.1 Motion Synthesis and Editing for Single Style Component

We test Algorithm 1 for each of the six style components. Usually 2 or 3 eigenvectors are required. For each test, the training phase takes about 30 s and it takes less then 0. 5 ms to synthesize motion. Fig. 7. 21 shows the results for style components "catty" and "stomp".

For Algorithm 2, it takes less than 0. 5 ms to edit an existing motion to a style degree designated by the user. Fig. 7. 22 shows the results on single style components "catty" and "rocky".

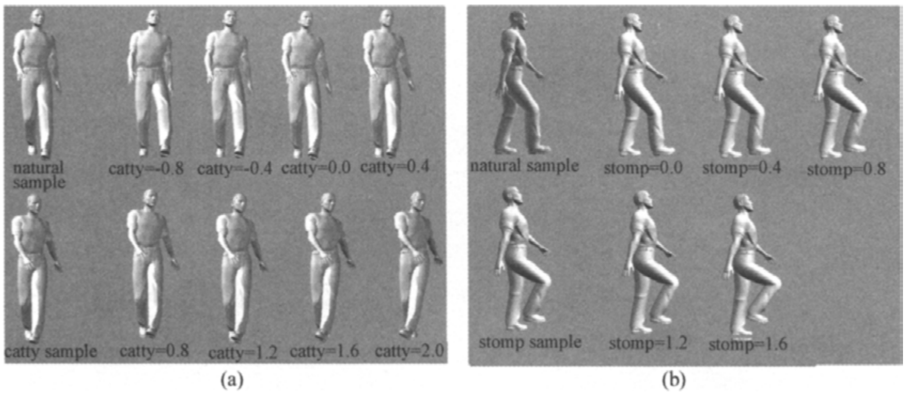


Fig. 7. 21 Algorithm 1 on single style component. (a) "catty"; (b) "stomp". The two motions on the first column are training samples randomly selected from the natural samples and catty samples for comparison. The eight motions on the right are synthesized by the algorithm

7.3.4.2 Motion Synthesis and Editing for Multiple Style Components

In order to testify Algorithm 3, we do some experiments on double style components. Usually 3 or 4 eigenvectors are preserved. It takes 30 s in the training stage. The synthesis process can be interactive. Typically it takes about 0. 5 to 2. 0 ms to synthesize a motion, depending on the number of iterations the algorithm executes. Fig. 7. 23 shows the result and comparison in the case of style [*catty*, *large-arm-swing*].

In Fig. 7. 23(a) every motion is showed by two representative frames in

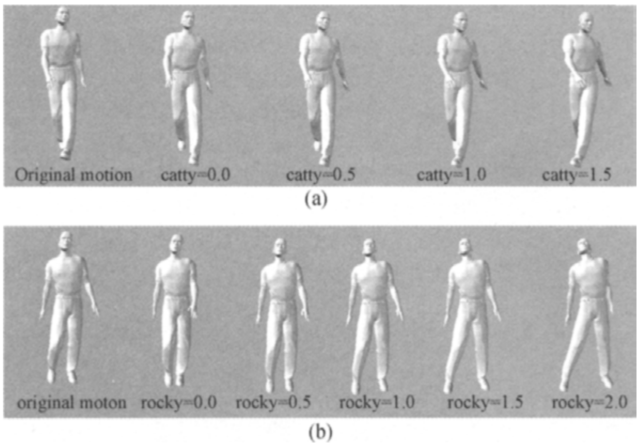


Fig. 7. 22 Algorithm 2 on single style component. (a) "catty". The first motion on the left is an existing motion to be edited. Note that it is already somewhat catty. The four motions on the right are returned by our algorithm for different catty degree designated by the user; (b) "rocky"

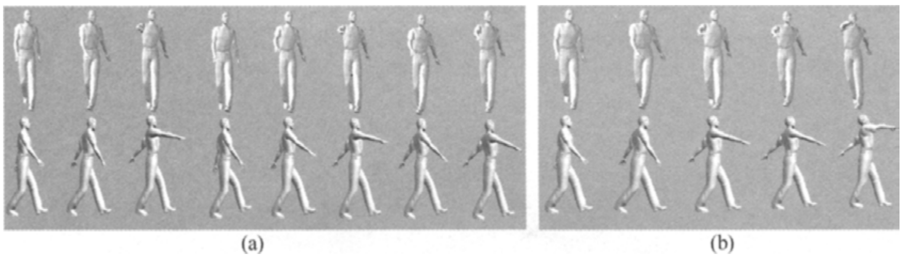


Fig. 7. 23 Algorithm 3 on style. (a) [*catty*, *large-arm-swing*]. Note that how our algorithm decouples the effects of the two style components while they are not orthogonal; (b) The result of simply dividing the problem into two separate single style problems

one column. The first three motions on the left are randomly selected from natural, catty and large-arm-swing samples for comparison. The five motions on the right are synthesized by the algorithm, with style description vector [0. 0, 0. 0], [0. 5, 0. 5], [0. 5, 1. 0],[1. 0, 0. 5] and [1. 0, 1. 0]. The views of motions on the first and second row give good ways for judging the catty degree and large-arm-swing degree, respectively. Note that the algorithm decouples the effects of the two style components and keeps the style description vector well quantified. This can be easily seen by some comparisons. For example, compare the motion with style description vector [0. 5, 0. 5] with that with [0. 5, 1. 0]. The two motions indeed have different "large-arm-swing" degree and mostly the same "catty" degree. Several similar comparisons can be made, convincing us that our algorithm successfully decouples the effects of the two style components, although they are not orthogonal. In the PCA space of this case, the angle between two styles' controlling vectors is about 35°.

Fig. 7. 23(b) compares our algorithm with the method of simply dividing the problem into two separate single style problems. From left to right, the first three motions are selected from natural, catty and large-arm-swing samples for comparison. The fourth motion is returned by Algorithm 3 for [*catty*, *large-arm-swing*] =[1. 0, 1. 0]. The last motion is derived simply by first synthesizing a motion with catty degree 1. 0 using Algorithm 1 and then editing the returned motion to large-arm-swing degree 1. 0 using Algorithm 2. Note that the motion returned by Algorithm 3 keeps the style quantified very well. The catty degree and large-arm-swing degree are the same as those of the catty sample and large-arm-swing sample, respectively. On the contrary, the last motion has poor style meaning. Its arm-swing degree is too large and its catty degree is insufficient. This is caused by the non-orthogonality of the two style components. When doing Algorithm 1 and Algorithm 2 separately, they cannot consider for each other so there are some severe side effects.

The editing process can be interactive. Typically it takes about 0. 5 to 2. 0 ms to edit a motion, depending on the number of iterations the algorithm executes. Fig. 7. 24 shows the result in the case of style [*high-step*, *large-arm-swing*]. Every motion is showed by two frames in a column. The first motion on the left is the original motion and the four motions on the right are the ones returned by Algorithm 4 for style description vectors [0. 5, 0. 5], [0. 5, 1. 0],[1. 0, 0. 5] and [1. 0, 1. 0] designated by the user. The views of motions in the first and second rows give good ways for judging the high-step degree and large-arm-swing degree, respectively. Note that the original motion has a large high-step degree and a very small large-arm-swing degree. The [0. 5, 0. 5] motion returned by the algorithm has the high-step degree mitigated and large-arm-swing degree strengthened. From [0. 5, 0. 5] to [0. 5, 1. 0], the large-arm-swing degree rises while the high-step degree remains. Similar observations can be made with other motions, which convince us that as in Algorithm 3, Algorithm 4 keeps the style description vector meaningful.

In this section we propose a framework for synthesis and editing human's walking motions on styles. We develop algorithms based on Principal Component Analysis (PCA). Using PCA, we reduce the data's di-

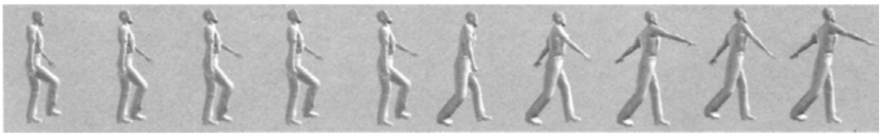


Fig. 7. 24 Algorithm 4 on style [high-step, large-arm-swing]

mension with a little information loss that is insignificant. The dimensionality reduction precludes large computational complexity. Usually it takes no more than one or two milliseconds to synthesize or edit a motion, so our algorithms can be employed for user-interactive systems, or large virtual environments in which large number of characters are required to be depicted quickly. We also extend our algorithms to the case when multi-dimensional motion style is required, which is of great use in practice. We illustrate the multidimensional-style algorithms with Algorithms 3 and 4 as examples, and the same idea can be extended to the cases with higher dimensional styles. We elaborate our whole framework with walking motions as examples, and this idea serves as well for other class of human motions such as running or jumping. And many motion style engines with different styles can be created by our method and embedded into animation system to synthesize and edit varied motion styles.

References

1. Grassia FS. Motion editing: mathematical foundations. In: SIGGRAPH 2000, ACM Press, 2000.
2. Gleicher M, Litwinowicz P. Constraint-based motion adaptation. Journal of Visualization and Computer Animation, 9(2): 65-94, 1998.
3. Gleicher M. Motion editing with spacetime constraints. In: Symposium on Interactive 3D Graphics, pp. 139-148, ACM Press, 1997.
4. Arikan O, Forsyth DA. Interactive motion generation from examples. ACM Transactions on Graphics (SIGGRAPH'02), 21(3): 483-490, 2002.
5. Li Y, Wang T, Shum HY. Motion texture: a two-level statistical model for character synthesis. ACM Transactions on Graphics (SIGGRAPH'02), 21(3): 465-472, 2002.
6. Lee J, Chai J, Reitsma PSA, Hodgins JK, Pollard NS. Interactive control of avatars animated with human motion data. ACM Transactions on Graphics (SIGGRAPH'02), 21(3): 491-500, 2002.
7. Schodl A, Szeliski R, Salesin DH, Essa IA. Video textures. In: SIGGRAPH 2000, pp. 489-498, ACM Press, 2000.
8. Eden AM. Directable motion texture synthesis. Technical Report, Harvard University, 2002.
9. Reynolds C. Flocks, herds, and schools: a distributed behavioral model. ACM Computer Graphics (SIGGRAPH'87), 21(4): 289-296, 1987.
10. Mataric M. Interaction and intelligent behavior. Technical Report: AITR-1495, Department of EECS, 1994.

11. Thalmann D. Virtual sensors: a key tool for the artificial life of virtual actor. In: Proceedings of Pacific Graphics 95, pp. 22-40, World Scientific Publishing, 1995.
12. Tu X, Terzopoulos D. Artificial fishes: physics, locomotion, perception, behavior. In: SIGGRAPH' 94, pp. 43-50, ACM Press, 1994.
13. Brogan DC, Hodgins JK. Group behaviors for systems with significant dynamics. Autonomous Robots, 4(1): 137-153, 1997.
14. Cormen YH, Leiserson CE, Rivest RL. Introduction to algorithms. The MIT Press, 1997.
15. Gleicher M. Motion path editing. In: SI3D'01, pp. 195-202, ACM Press, 2001.
16. Jolliffe IT. Principal component analysis. Springer Series in Statistics. Springer-Verlag, 1986.
17. Kovar L, Schreiner J, Gleicher M. Footskate cleanup for motion capture editing. In: ACM/EuroGraphics SCA 2006, pp. 97-104, ACM Press, 2002.

8

Intelligent Techniques for Character Animation

Motion capture based animation has become one of the most promising areas in computer animation. But the motion capture equipment is expensive. To be able to reuse the data and make new movements, we need to modify or edit the motion capture data before retargeting it to animated characters.

Recently, extensive study has been done on these issues, which basically consists of three motion-editing techniques. (1) Movement curve fitting and control point adjusting method [1]. This method is not suitable when the movement is big. (2) Data signal processing [2,3]. This method promotes the reuse of existing data, for example, motion warping presented by Witkin and Popvic [4] uses blending and overlapping algorithms to generate new motion based on existing motions. Bruderlin and Williams [5] regarded motion as signal. Signal-processing techniques such as multi-resolution filter, time warping, multi-target motion interpolation, motion wave-shaping and motion displacement mapping are used to regenerate motions based on captured motion data. (3) Spatio-temporal constraint method [6,7], is suitable for generating interactive motion. It generates constrained movements by solving objective movement equations [6,8,9]. Using this method, Popvic and Witkin [10] transformed captured motions into new motions that preserve the original properties of motion and satisfy users' specifications by solving dynamic equations. Rose, et al. [11] managed to generate the transition of two clips of motion. Lee and Shin [12] generated constrained motions using curve fitting and inverse kinematics. But each instance requires its unique equations and solutions, and it is hard to generalize.

The above-mentioned methods have largely solved the problem of data reuse and motion regeneration. The problem is that they can only handle one character, while in reality, multiple animated characters move simultaneously in one scene. The spatio-temporal constraint, imposed by the virtual environment and other characters, also changes constantly. Because the original motion is captured in a structural environment, before the data is retargeted to animated characters in a complex non-structural environment, motion editing is always needed. We think it is important to enable the automatic perception of virtual scenes and the collaborative work of multi-characters.

In Sect. 8. 1, we present the idea of motion fusion of multiple animated characters and the approaches to fuse multiple motions into one non-structured virtual scene by improving the perception and self-decision-making abilities of characters. Our basic idea is as follows. Firstly, motion decision-making and motion collaboration are applied to perceive the virtual environment and other moving characters automatically. Secondly, a Definite State Machine (DSM), dealing with motion mode and spatio-temporal constraints imposed by the ambient scene and other moving characters, is utilized to select a suitable motion mode for each character intelligently. Finally, for each character, available spatio-temporal constraints can be applied to solve continuous movement based on constraints imposed by other characters, ambient virtual scene and animators. Different from available spatio-temporal constraints, our methods automatically perceive the spatio-temporal constraints and improve the perception and decision-making of each animated character.

In Sect. 8. 2, we propose a framework of script engine for realistic character animation based on Motion Capture (MoCap) database. Our script engine produces realistic human animation in either offline mode or online mode. In offline mode, users make motion scripts, describing the motion properties, such as type, order, and details, by a common text editor or a graphical user interface provided by our system. Then the scripts are decomposed into sequential commands for retrieving relevant motion clips from a MoCap database and generating final animation sequence. In online mode, users employ keyboard, mouse or other peripheral input equipments to activate motion commands and obtain synthesized movement sequence in real-time. Furthermore, users can define their own motion elements table and scripts format suitable for various MoCap datasets. In experiments, we built two applications based on this script engine and verified their capability on two different MoCap datasets respectively. The experimental results show that this script engine framework can produce robust results and be used as a human motion engine in various applications, such as computer games, movies, sports simulation and virtual reality.

In Sect. 8. 3, we propose a framework to program the movements of characters and generate navigation animations in virtual environment. Given a virtual environment, a visual user interface is provided for animators to interactively generate motion scripts, describing the characters' movements in this scene and finally used to retrieve motion clips from MoCap database and generate navigation animations automatically. This framework also provides flexible mechanism for animators to get varied resulting animations by configurable table of motion bias coefficients and interactive visual user interface. This method is more alike to make a program in a visual development environment, and therefore named "motion programming".

8. 1 Multiple Animated Characters Motion Fusion

8. 1. 1 Architecture of Multiple Animated Characters Motion Fusion

We assume that human movement is a four-layer thinking-and-moving process. The process of animated characters movement is analogous to human movement, as shown in Table 8. 1. Motion decision-making, motion collaboration, motion solving and motion execution are directly next to implement motion planning, collaboration, solving and displaying. Based on the analogy between human movement and animated character movement analyzed above, the general architecture of multi-character motion fusion is conceived as consisting of four parts as illustrated in Fig. 8. 1.

Motion decision-making layer First, modeling virtual scenes and writing the scripts of producing animation, then planning multiple animated characters' paths in non-structured virtual scenes and breaking these paths into single character's path.

Motion collaboration layer Realizing the character-to-character collaboration and character-to-environment collaboration by settling two basic problems: decision-making of discrete motion mode, and generating spatio-temporal constraints imposed by virtual scenes and other characters.

Motion-solving layer First, transforming the captured motion and obtaining the approximate pose at each instance for each character, then resolving the continuous motion of every discrete motion mode.

Table 8. 1 The analogy of human movement and animated characters movement

Layer	Motion of human	Motion of animated character
Motion decision-making	The path and behaviour are decided by the brain	Animator plans the routine and process of animated character movement
Motion collaboration	Breaking motion decision into units, and collaborate with ambient scenes	Computer collaborates the relation among characters and environment, and achieves the discrete motion strategy in each part of the path
Motion solving	The specific pose at one time instant is solved	Computer solves the continuous movement of each animated character
Motion execution	Each part of human executes the solved motion	Motion data drive the animated characters to move

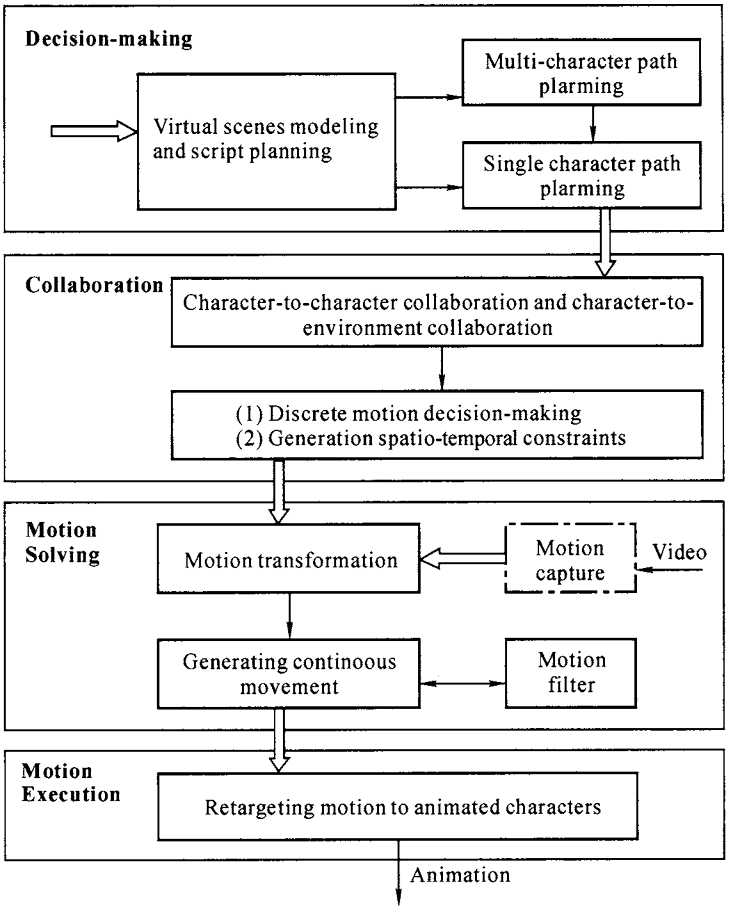

Decision-making
Virtual scenes modeling and script planning
Multi-character path plarming
Single character path plarming
Collaboration
Character-to-character collaboration and character-to-environment collaboration
(1) Discrete motion decision-making
(2) Generation spatio-temporal constraints
Motion Solving
Motion transformation
Motion capture
Video
Generating continoous movement
Motion filter
Motion Execution
Retargeting motion to animated characters
Animation

Fig. 8. 1 General architecture for motion fusion

Motion execution layer Retargeting the regenerated continuous movement to different animated characters in a complex non-structured environment, and producing one clip of vivid animation of multiple animated characters moving collaboratively and simultaneously.

In the study of fusing multi-characters motions, there are two main challenges: (1) the animated characters' ability of self-determination to select the optimum path in non-structured virtual environment is limited; (2) the decision-making strategy of the discrete motion mode of each animated character and the solving of continuous movement along a specific path are difficult to implement. Discrete motion mode means the preferential motion mode, such as walking, running and jumping, selected by one character moving on a specific path, while continuous movement means a sequence of poses for one character moving on a specific path in a period of time. Here we apply the idea presented by Kalisiak to path planning for multiple charactėrs. A more thorough introduction can be found in [13].

In the later discussion, we will give the idea and approaches to fuse multiple animated characters' motion into one scene and show the experimental results.

8.1.2 Collaboration of Multiple Animated Characters

When we fuse multiple motions into one scene at the same time, various characters should collaborate with each other in thought and action, and work without collision. On the other hand, a complex virtual environment always imposes various constraints on the characters; thus each character must also collaborate with the virtual environment. As a rule, in motion fusion of multiple animated characters, collaboration includes character-to-character collaboration and character-to-environment collaboration. In the study of motion fusion, the key issues relating to motion fusion are motion collision detection and avoidance, motion mode selection for each character moving on the specific paths, analysis and specification on spatio-temporal constraints imposed by virtual environment and other characters moving in the same scene, and so on.

8.1.2.1 Collaboration between Characters and Virtual Environment

Nowadays, most commercial software devoted to computer animation and games runs in a structured virtual environment. When augmenting the vividness and diversity of the virtual environment, we need to make a further study of the effects on character movement resulting from the diversity and complexity of the virtual scene as well as the bumpiness and stochasticity of planned paths.

Generally, there are several steps in the collaboration between animated characters and the virtual environment. First, spatio-temporal constraints imposed by the environment are analyzed and specified. Second, by taking the spatio-temporal constraints as input, a DSM is applied to deduce the motion mode of animated characters satisfying the spatio-temporal constraints. Some of the constraints imposed by the virtual scene are specified through a human-computer interface; others are detected and specified by our system automatically.

8.1.2.2 Collaboration among Characters

Collaboration in Action

Collaboration in action consists of active collaboration and passive collaboration. Active collaboration means that various animated characters, like autonomous agents, readily react to the ambient environment and other characters. For example, if one character (A) finds another one (B) at time instant t, A will make a self-decision of whether to shake hands with B or not. Passive collaboration means that the animator specifies the action of animated characters in advance in order to accomplish one assigned task. For example, the animator specifies that two characters accomplish hand-shaking in a specific position at time instant t. In essence, the collaboration between animated character and environment can be regarded as the collaboration between active character and immobile character. Thus we can apply a DSM to deduce the mode of motion in a specific path, and then solve continuous movement by spatio-temporal constraint approaches and the numerical optimum method.

Collaboration in Locomotion

Except for the action collaboration among multiple animated characters, multi-character motion fusion should guarantee the collaboration of various characters in motion. As an example, let us consider the situation of two characters moving on the same path, where they may collide with each other; or the situation of passing through a bridge which actually only allows one person through at a time. This bridge is regarded as an exclusive resource in a computer technology. If two characters want to grab an exclusive resource synchronously, a deadlock will occur. Therefore, before solving continuous movement for each animated character, we need to detect the probability of collision of various characters and deadlock resulting from grabbing exclusive resources.

Collision among multiple animated characters As a rule, any two rigid human bodies should not penetrate each other. To avoid collision before deadlock occurs, motion simulation is applied to detect collision. Once collision has been detected, motion mode will be updated to avoid collision accordingly.

Most of the presented approaches to detect collision are reasonable but complicated to implement. In motion fusion of multiple animated characters, we assume that collision takes place when the Euclidean distance between character m and character n is less than a maximal threshold $d_{\max}$, when $\| \boldsymbol{q}_m(t) - \boldsymbol{q}_n(t) \| \leqslant d_{\max}$. Then, we calculate the distance between one joint in one character and a triangular mesh composed of joints in the other character, and detect the point of intersection. If the point of intersection is found, it means that collision will occur in the process of movement; otherwise, two characters will work smoothly. Based on the knowledge of physiology, the maximal threshold can be regarded as approximately half of the height of the human model.

Once collision has been detected, there are different strategies to avoid it: (1) maintain the former speed, update the direction of movement; (2) maintain the former direction, update the relative speed to other characters; (3) change both the speed and direction.

Management of exclusive resources The commonly used approach to control exclusive resources is by using tokens. But this cannot guarantee efficiency, and therefore does not fit for managing non-exclusive resources. In a collaborative virtual environment, different characters have different authentication to apply to and operate on an exclusive resource. Therefore this approach cannot distribute an exclusive resource equally. In order to overcome the shortcomings of approaches using tokens in a multiple animated characters motion fusion, we apply a two-layer management strategy to control share of exclusive resources during characters' movements: (1) detect all exclusive resources in the virtual environment, and then detect the characters using exclusive resource; (2) based on the detected results, we control concurrently the characters' movement using preference.

8.1.2.3 Discrete Motion Decision-Making

In the character-to-character collaboration and character-to-environment collaboration, a character's movement is constrained by the ambient scene. From the viewpoint of vividness and naturalness, in certain phases of the path, different reasonable motion modes should be selected. Therefore, depending upon the constraints imposed by other characters and the virtual environment, preferential motion modes suitable for specific phases of the path, such as walking, running, climbing and jumping, should be selected before solving continuous movement.

The discussions above are mainly to perceive the spatio-temporal constraints imposed by the ambient virtual environment and other moving characters. In the later discussion, we need to make reasonable decisions based on these constraints in order to help the virtual characters to select preferable motion modes. In fusing motions of multiple virtual characters, a DSM is introduced to make intelligent decisions regarding motion modes for the virtual characters. Creating a database of constraints and a database of characters' behavior is essential for intelligent decision-making. The database of constraints covers the constraints imposed by the ambient virtual scene, such as barrier, tunnel and threshold, and the constraints imposed by other moving virtual characters, such as shaking hands, nodding, stooping down and standing aside. The database of behavior covers all possible motion modes in the non-structured virtual environment.

To select a preferential motion mode from various types of motions, a DSM is introduced to solve discrete motion. We introduce DSM not only as a method to select the motion mode, but also as a breakthrough in traditional computer animation directed by an animator. By using a DSM, we implement an active collaboration methodology in computer animation, which enables characters to perceive the ambient environment, to make self-decisions of action and movement according to the constraints imposed by the ambient scene and other characters.

A DSM for selecting the motion mode can be depicted as a set (Q,Σ,δ)

composed of triple elements. Q is a definite set of states in which each element is one type of motion mode, Σ is a finite set of inputs in which each element is a spatio-temporal constraint imposed by other characters or virtual scene. δ: $Q \times \Sigma \to Q$ is the conversion function of states, which means that when character is in state q_0 ($q_0 \in Q$), if the constraint ω ($\omega \in \Sigma$) is input, the DSM will output a definite state q_1 ($q_1 \in Q$).

An example of DSM as shown in Fig. 8. 2 is used to explain how it realizes discrete motion decision-making. Let the motion mode be $Q=$ {skipping, walking, running, nodding, spanning, handshaking}. If the motion mode for one character is $q=$walking, spatio-temporal constraints imposed by the virtual environment and other characters are *con*, satisfying $con \in \Sigma' = \{a_0, a_1, a_2, a_3, a_4, a_5\}$. The mode $q_1 \in Q' =$ {skipping, running, nodding, spanning, handshaking} will be selected. For example, if *con* is meeting people, the motion mode of character will be changed to handshaking. When *con* encounters a trap, the motion mode of character will be changed to spanning.

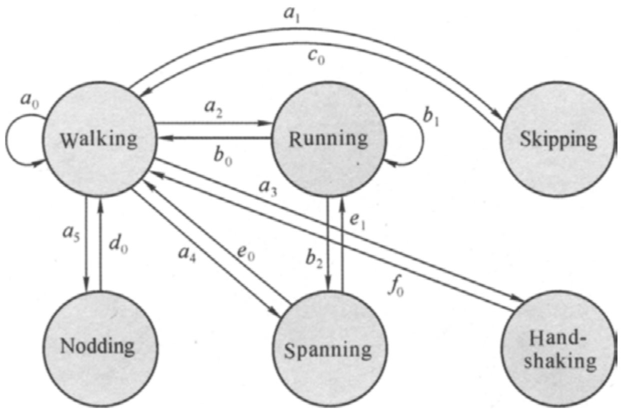


Fig. 8. 2 An example of DSM

8. 1. 3 Solving Continuous Motions

8. 1. 3. 1 Approximating Continuous Motions

To enable animated character moving on the planned path in a virtual environment which differs from the realistic one, we transform the captured motion to make the character pass the planned path first and get the approximate pose at each instance for each character moving in the new scene, which simplifies the process of solving continuous movement of each character. We use Euclid angle and translation vector to represent the movement sequences of the animated character. The original motion is then transformed in order to cover the planned path using cycle extension and displacement mapping.

Human Model

The human model adopted in the video-based motion capture system developed by our lab is different from the markers and special reflective objects adopted in Fua, et al.[14], which are costly and inconvenient. We have designed a suit of tight clothing. At each joint of the tight clothing there is a block with a unique color. Compared with special markers or sensors appended to the body, it has several advantages: (1) it is inexpensive, easy to implement and convenient to track; (2) it is free from constrained movement. Based on the tight clothing, we have defined the corresponding human body as a set of rigid body parts connected by joints, and human motion as the movement of the human skeleton. Fig. 8. 3 is the adopted skeleton model, which consists of 16 joint points. Our target of motion capture is to extract the 3D human body movement sequences of each color block's center on the tight clothing.

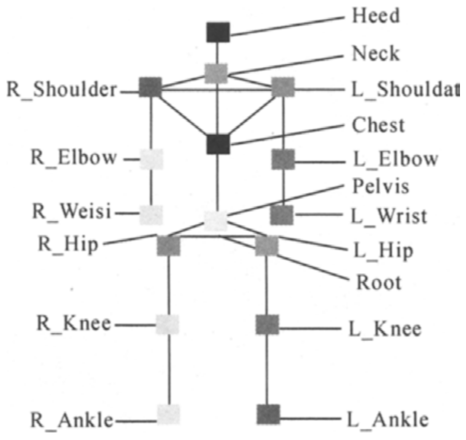


Fig. 8. 3 Human model

Motion Sequence

Depending on motion capture data and regarding human motion as rigid body movement, we can represent the motion sequence $\boldsymbol{Q}(t)$ as:

$$\boldsymbol{Q}(t)=(\boldsymbol{q}(t),p_0(t),\cdots,p_n(t))=(\boldsymbol{q}(t),\boldsymbol{p}(t)) \qquad (8\text{-}1)$$

where $\boldsymbol{q}(t)$ is the translation vector of point Root (as seen in Fig. 8. 3) at time t, and $p_i(t)$ is the rotation angle of joint i at time t.

Cycle Extension

The frame number of captured motion may be too small to be used directly in an animated character of a new environment; therefore, cycle extension should be used to extend the cycles of motion. If the movement sequence of original motion in cycle T is $\boldsymbol{Q}(t) \sim \boldsymbol{Q}(t+T)$, the orientation to extend will be $(\boldsymbol{p}(t+T)-\boldsymbol{p}(t))/\|\boldsymbol{p}(t+T)-\boldsymbol{p}(t)\|$. The corresponding movement of n cycles will be depicted as:

$$\boldsymbol{Q}'(t)=\boldsymbol{Q}(t)\cdot(n(\boldsymbol{q}(t+T)-\boldsymbol{q}(t)),(\boldsymbol{p}(t+T)-\boldsymbol{p}(t))/\|\boldsymbol{p}(t+T)-\boldsymbol{p}(t)\| \tag{8-2}$$

Displacement Mapping

Since the start point and direction of movement of different paths are different from the original motion, displacement mapping should be applied to transform the original motion. Let original motion be $\boldsymbol{Q}(t)$ and transformed motion be $\boldsymbol{Q}'(t)$ after applying displacement mapping to $\boldsymbol{Q}(t)$. Let the displacement vector from $\boldsymbol{Q}(t)$ *to* $\boldsymbol{Q}'(t)$ be $(\boldsymbol{U}(t),\boldsymbol{V}(t))$. $\boldsymbol{U}(T)$ and $\boldsymbol{V}(t)$ are translation vector and rotation angle at time instant t, respectively. $\boldsymbol{T}(t)$ is the translation vector from $\boldsymbol{Q}(t)$ to the original point of the world coordinate system at time instant t. Displacement mapping can be depicted as:

$$\boldsymbol{Q}'(t)=\boldsymbol{Q}(t)\oplus(\boldsymbol{U}(t),\boldsymbol{V}(t)) \tag{8-3}$$

We can work out $(\boldsymbol{U}(t),\boldsymbol{V}(t))$ depending upon the planned paths and calculate the movement sequences which make animated characters move on the planned paths using equation (8-3).

A few artificial motion properties, such as slipping and two legs hanging in the air, will be involved in the initial motion resulting from above motion transformation. The reason for these artifacts is that some essential spatio-temporal constraints are not imposed by the ambient virtual scene or animators. Therefore, we need to apply motion collaboration among the characters and the virtual environment to produce enough essential constraints for solving natural continuous movement for each character in the following subsection.

8.1.3.2 Solving Continuous Motions Based on Spatio-temporal Constraints

In a scene, specific spatio-temporal constraints should be imposed for multiple animated characters to cooperate with each other at the same time. The aim of a spatio-temporal constraint method presented by Gleicher[8] is to produce motion, which satisfies spatio-temporal constraints and minimizes objective function of movement. Generally, spatio-temporal constraints specify what a character will act in a certain space and time, while objective function specifies how to accomplish this motion. Continuous motion solving produces a dynamic motion, which satisfies both the specified spatio-temporal constraints and objective function.

Spatio-temporal Constraints

Spatio-temporal constraints are special restrictions on a character's movement imposed by the virtual environment and other characters in time and space. It specifies what the character does, which makes the character move depending upon ambient circumstance but preserves the vividness and inherent properties of motion. For spatio-temporal constraints, it can be specified dynamically or Kinematically. Dynamically, the constraints to be considered are very complex. Therefore, only kinematic spatio-temporal constraints are generally considered so as to limit the movement of character in space and time domains. Typically, spatio-temporal constraints are described as a set of equations. In continuous motion resolving, differential constraints are specified during a period of time, and they can be transformed to the constraints at one time instant. We solve each frame to satisfy constraint functions and objective function in movement sequences.

Spatio-temporal constraints are commonly described as a set of constrained equations, which can be described as:

$$\boldsymbol{F}(\boldsymbol{q}(t), \boldsymbol{p}_0(t), \cdots, \boldsymbol{p}_n(t)) = \boldsymbol{C} \tag{8-4}$$

where t is the time instant of character movement, $\boldsymbol{q}(t)$ and $\boldsymbol{p}_i(t)$ are the parameters of character movement and $\boldsymbol{C}$ is a constant.

Major spatio-temporal constraints used in continuous movement resolving include:

- Position constraints: $\boldsymbol{q}(t) = \boldsymbol{C}_0$, where $\boldsymbol{C}_0$ is the position constant;
- Pose constraints: $\boldsymbol{p}_i(t) = \boldsymbol{C}_1$, where $\boldsymbol{C}_1$ *is the Euclid angle constant*;
- Position constraints supporting animated character body: $\boldsymbol{q}_i(t) = \boldsymbol{C}_2$, where i is joint i and $\boldsymbol{C}_2$ is the position constant.

Objective Function of Movement

Objective function concerns how to accomplish a motion. In resolving continuous motion, after specifying objective function for character movement, the system is able to select an exclusive solution from a set of reasonable solutions. In general, it is difficult to get a perfect objective function, and users directly use constraints as objective function, the interactive performance of which enables the user to make adjustments by adding more constraints. Gleicher[8] pointed out that there are some drawbacks to simply minimize the magnitude of the parameter vector. One particular problem is that different parameters often have vastly different effects. To overcome these problems we use a weighted sum-of-squares of the parameters. Our objective can be seen as an approximation to the function that minimizes the displacement from the ideal position.

In motion fusion, two major objective functions are as follows.

- Minimizing the difference of position between ideal joints and practical joints, which can be described as:

$$\min\left(sum = \int_{t_{\text{start}}}^{t_{\text{end}}} W(t,k) \sum_{k=1}^{16} \| \boldsymbol{p}_k(t) - \hat{\boldsymbol{p}}_k(t) \|_2 \mathrm{d}t\right)$$

  where $\boldsymbol{p}_k(t)$ is the coordinate of joint k after motion fusion, $\hat{\boldsymbol{p}}_k(t)$ is the coordinate of joint k before motion fusion, t_{start} and t_{end} is the start and end in the region of motion edition, respectively, and $W(t,k)$ is a weighted factor of joint k at time t.
- Minimizing the difference of movement before and after motion editing, which can be described as:

$$\min\left(sum = \int_{t_{\text{start}}}^{t_{\text{end}}} W(t) \| Q(t) - Q'(t) \|_2 \mathrm{d}t\right)$$

  where $Q(t)$ is the pose after motion editing, $Q'(t)$ is the ideal pose, and $W(t)$ is a weighted factor.

Re-solving Continuous Movement

The collision and constraints in motion fusion are described as a set of equations; therefore, we can view the motion fusion problem as constrained numerical optimization. Non linear equations cover potentially large numbers of constraints and variables since we create a single problem for multiple motions, which makes solving multi-character motion fusion all the more challenging.

For simplicity, we can build approximations of the nonlinear problem for solving them. Iteration algorithms and initial solution, discussed above, are used to obtain an exact solution. We use the algorithms discussed in [8], which belong to a class of sequential quadratic programming that has linear constraints and quadratic objective functions. Moreover, only equality constraints are considered in our solver.

8.1.4 Motion Rectification

The solved motion after discrete motion decision-making and continuous motion solving reveals a few limitations: (1) jitter resulted from linear interpolation, which lacks vividness and naturalness; (2) bumpiness of motion trajectory was because continuous motion solving does not consider the smoothness of the trajectory; (3) an unnatural motion mode was selected for some characters. Therefore, motion rectification, namely the smoothing of the solved motion, must be applied.

Filter of Motion Trajectory From the discussion above, we know that the translation vector of motion is composed of a set of discrete moving points. We adopt the following motion filter method: fit the discrete points, and then generate the curve of motion, which minimizes the difference between the specified trajectory and the solved curve. The generated continuous curve is then used to update the translation vector of motion sequence.

Imposing Constraints of Feasible Motion Usually, it is impossible to create transitional motion by linear interpolation. For instance, the body may hang in the air. We re-impose feasible constraints on the motion and use inverse kinematics to solve the position of each joint for vivid motion.

Filter of Limb Movement By using a Kalman filter, we can remove noise and overcome the inconsistency of limbs resulting from the above methods.

8.1.5 Results and Discussion

8.1.5.1 Results

Based on the discussion in the previous sections and other research[15], our lab has developed a Video-based Human Animation (VBHA) system with two cameras. It is implemented with Visual C++ and can run on the Windows X platform. Its main functions include calibration, feature tracking, 3D reconstruction, motion editing, and multiple animated character motion fusion and production animation.

In Fig. 8.4, we demonstrate the main ideas of multi-character motion fusion. The motion trajectories of four characters planned by the animator

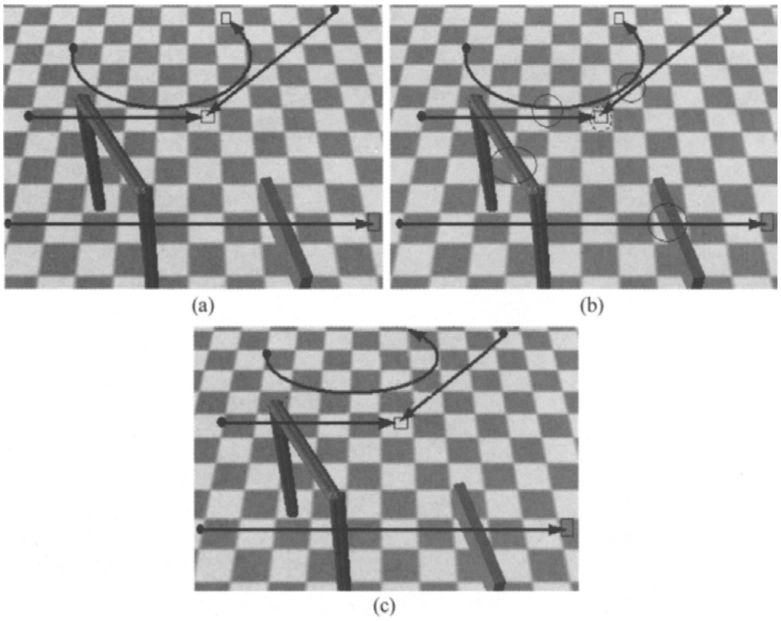


Fig. 8.4 Process of motion collaboration and solving

are shown in Fig. 8. 4(a); the spheres denote starting points and the rectangles denote ending points of animated characters. Fig. 8. 4(b) shows four possible collisions of multiple animated characters' motion detected by the system (denoted by a circle) and the functions specified by the animator (denoted by a dashed circle). Discrete motion modes and constraints, which result from motion collaboration and collision detection, are transported to the motion-solving layer. The trajectory of continuous movement solved by our system can be seen in Fig. 8. 4(c).

To test the feasibility and efficiency of the multi-character motion fusion approach presented in this section, we fuse multiple motions of animated characters into one scene with captured motion data. Fig. 8. 5 shows the experimental result that fuses motions of four characters into one non-structured virtual environment with a door and a barrier. Animator specifies initial motion trajectories of each character and passive collaboration, such as handshaking, while the system automatically selects the motion mode and produces continuous movement of each character. In the motion collaboration layer, once having detected the door in the scene, the system selects the motion mode depending on the relative height of the character and the door. Similarly, when the barrier is detected, the system makes the decision of spanning the barrier independently. In the motion-resolving layer, continuous movements of each character are generated based on spatio-temporal constraints and motion mode. From Fig. 8. 5, we can see that feasible discrete motion mode and vivid continuous movement can be generated by our system.

Analysis of the results of our system reveals three limitations of fusing the motions of multiple virtual characters.

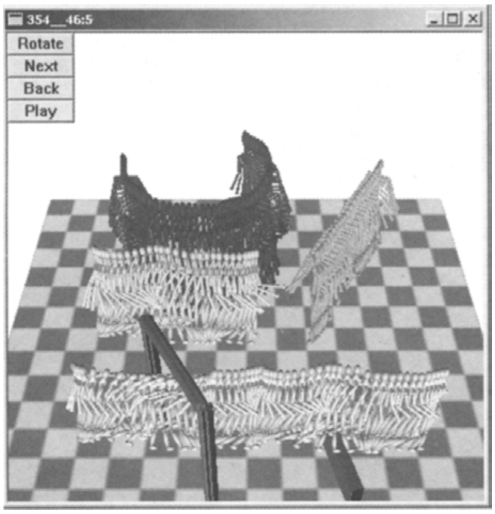


Fig. 8. 5 Multiple animated characters motion fusion

- First, it is essential for our system to save the 3D position of the virtual environment beforehand. Unlike the human, our system cannot reconstruct a 3D virtual environment automatically. Consequently, a complex virtual environment is still not practical.
- Second, a DSM can make an intelligent decision of motion mode. But a perfect database of constraints and behavior is required, which is difficult to obtain. Moreover, once a suitable rule of DSM for some character is not found in the database, DSM will fail. As the result, we think that machine learning should be introduced to improve the decision-making of DSM.
- Finally, in fusing the motions of multiple virtual characters, our system has not finished planning the path for each character automatically. At present, we specify the path for each character manually.

8.1.5.2 Discussions

This section presents a new idea of multi-character motion fusion and its relevant implementation approaches. By fusing motions captured in a structured environment into one non-structured virtual environment, we have studied and implemented the collaboration between characters as well as the collaboration between character and virtual environment. The DSM method, adopted in solving discrete motion mode, enhances the perception and self decision-making of the virtual characters as well as improving the automation degree of multiple animated characters working cooperatively in complex circumstances. Moreover, motion transformation presented in this section improves the reusability of motion capture data and simplifies the process of solving continuous movement by redirecting original motion data to a new path in a virtual environment through cycle extension and displacement mapping. High automation and high reusability make our presented approach highly applicable in computer animation and computer games.

Further studies on our multi-character motion fusion approach should be concentrated on the following. (1) In discrete motion mode decision-making, the number of rules in DSM determines the correctness of reasoning. A better decision-making capability should be studied for a complex environment. (2) In motion fusion, constraints imposed by a complex virtual environment increase as the number of the characters increase, and computation complexity rises. Further research is expected to improve the efficiency of the algorithm and lower the computation complexity. (3) There is a trend for studying autonomous cooperation among several characters in the computer animation field. We should strengthen the artificial life science application in computer animation.

8.2 A Script Engine for Realistic Human Motion Generation

Due to the popularity of optical motion capture system, more and more realistic human motion data can be acquired easily. In recent years, large and highly detailed human motion database is commercially available and widely used in various applications such as video games, animation films, sports simulations and virtual reality. Therefore, many researchers have focused on how to edit, manipulate and reuse the existing motion data.

In practical situations, most of the time, users need an engine (or application) to produce long, complex and controllable human movement sequence directly and efficiently. And this engine (or application) should meet the following requirements:

- It should be a high level approach and focus on producing long, complex and controllable human movement sequence instead of a short motion clip;
- It can be encapsulated as an engine and used in various applications, such as video games, animation production, sports simulations and virtual reality;
- It should be extendable and flexible for adapting various kinds of human motion dataset, such as simple locomotion and complex fighting motions;
- By this engine (or application), the generated results should be reusable for further editing or improvement;
- It should have multiple user interfaces, such as normal text typing, GUI or peripheral input equipments, and can be used easily and intuitively.

But to our knowledge, there are very few approaches or tools which have been developed to meet the above-mentioned requirements. So in this chapter we propose a framework of script engine based on well-organized MoCap database to achieve this goal.

Firstly, motion dataset, which contains some realistic and standard human motion clips for special application (such as some simple locomotion for avatar navigation in virtual reality, or a set of fighting motion clips for video games), is obtained by optical motion capture system or skilled animators. For each motion dataset, every motion clip is well-segmented, annotated and stored in motion database. Then users can generate or edit motion scripts (or script commands) by common TXT editor, graphical user interface or some peripheral input equipments (such as keyboard, mouse or joystick, etc). In movement generation processing, motion scripts are decomposed into sequential commands which are used to retrieve proper motion clips from motion database. Then the retrieved motion clips are assembled into final realistic human movement sequence. Accounting to the incremental motion datasets for some special applications, users can define their own motion element table for motion script and parser. What's more, the script used in this framework is defined as XML style which is convenient to reuse, extend and manipulate. Fig. 8. 6 gives the workflow of our framework.

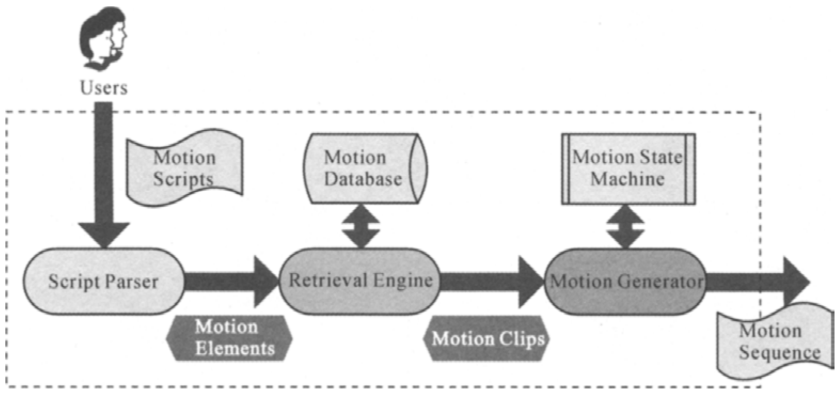


Fig. 8. 6 The workflow of our framework

8. 2. 1 Motion Database Setup

Logically, the whole motion database can be divided into several datasets and each dataset contains some standard motion clips which can be reused in different applications. For example, a locomotion dataset used for avatar navigation in virtual reality contains standard walk, run, and jump motion clips, a fighting dataset used for game development contains punch and kicking motions, and a sports dataset used for gymnastics simulation contains some standard gymnastic actions, etc. Fig. 8. 7 gives the logical architecture of motion database.

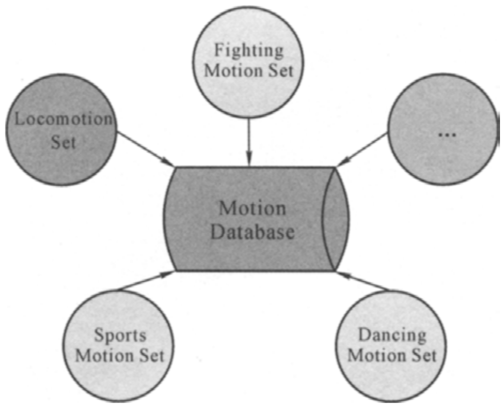
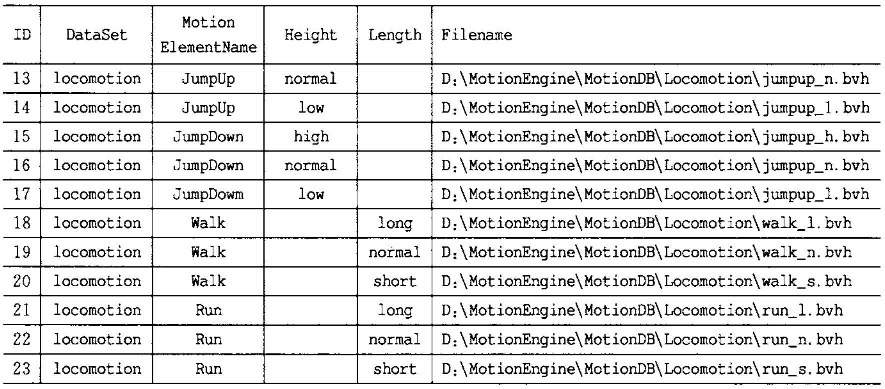


Fig. 8. 7 Logical architecture of motion database

After motion capture, the original TRC ① motion data are converted into BVH mo-

① TRC: The original motion data file from Motionanalysis optical motion cupture system.

tion files and each motion clip is well-segmented, annotated and stored in motion database. For the periodical movements, only one complete cycle is stored. Fig. 8. 8 gives a snapshot of our motion database.



ID	DataSet	Motion ElementName	Height	Length	Filename
13	locomotion	JumpUp	normal		D:\MotionEngine\MotionDB\Locomotion\jumpup_n.bvh
14	locomotion	JumpUp	low		D:\MotionEngine\MotionDB\Locomotion\jumpup_l.bvh
15	locomotion	JumpDown	high		D:\MotionEngine\MotionDB\Locomotion\jumpup_h.bvh
16	locomotion	JumpDown	normal		D:\MotionEngine\MotionDB\Locomotion\jumpup_n.bvh
17	locomotion	JumpDowm	low		D:\MotionEngine\MotionDB\Locomotion\jumpup_l.bvh
18	locomotion	Walk		long	D:\MotionEngine\MotionDB\Locomotion\walk_l.bvh
19	locomotion	Walk		normal	D:\MotionEngine\MotionDB\Locomotion\walk_n.bvh
20	locomotion	Walk		short	D:\MotionEngine\MotionDB\Locomotion\walk_s.bvh
21	locomotion	Run		long	D:\MotionEngine\MotionDB\Locomotion\run_l.bvh
22	locomotion	Run		normal	D:\MotionEngine\MotionDB\Locomotion\run_n.bvh
23	locomotion	Run		short	D:\MotionEngine\MotionDB\Locomotion\run_s.bvh

Fig. 8. 8 Table in motion database

The "DataSet" column contains dataset name of each motion clip, it could be "locomotion", "fighting motion", or "sports motion", etc. And the users can create their own motion dataset and corresponding motion table. The "MotionElementName" column contains the motion element name which means the content of motion clip and corresponds to the "type" attribute of ⟨Motion⟩ tag in the motion script. "Height" and "Length" columns contains the physical properties of each motion clip, such as "high" property of "JumpUp" motion means that this motion clip is a jump-up movement with big height.

In this research, we set up a testing database that contains two datasets, "locomotion" and "fighting", for experiments. There are totally 25 standard motion clips in "locomotion" dataset and Table 8. 2 gives the motion clip list of this dataset.

Table 8. 2 Motion clips in "locomotion" data set

Motion Element Name	Height			Length		
	high	normal	low	long	normal	short
Walk	—	—	—	1	1	1
Run	—	—	—	1	1	1
Stepup	1	1	1	—	—	—
Stepdown	1	1	1	—	—	—
Stepover	1	1	1	—	—	—
Jumpup	1	1	1	—	—	—
Jumpdown	1	1	1	—	—	—
Jumpover	1	1	1	—	—	—
Stand	1					

8.2.2 Motion Script

By our framework of script engine, users can make motion scripts by common TXT editor or some other interactive approaches, such as graphical user interface and peripheral input equipments. Actually, the motion script consists of some sequential tags which define the motion clips with specified physical properties. When the motion scripts are passed to the script engine, they are decomposed into divided commands to generate movement sequence.

In our script engine, motion script is made and stored as XML style, which is a widely used markup language standard and convenient to reuse and extend. Fig. 8.9 gives an example of motion script segment and corresponding DTD file designed for the "locomotion" dataset. In the definition of DTD file in Fig. 8.9(b), we can see that the motion script consists of a group of 〈Motion〉 tags with different attributes. The attribute "type" corresponds to the motion element in the "locomotion" dataset. The attributes "height" and "length" mean the physical properties of motion clips. By combining the attributes of "type", "height" and "length", a specific motion clip can be indexed. The attribute "repeat" means the repeat number of this motion clip in movement sequence. So the motion script in Fig. 8.9(a) can be interpreted as follows:

- The character starts with a "stand" motion;
- Follows 3 cycles of "walk" movement with normal step-length;

```
〈? xml version="1.0" encoding="UTF-8"?〉
〈! DOCTYPE MotionSequence SYSTEM" me.dtd"
〈MotionSequence〉
    〈Motion type="stand" repeat="1"〉〈/Motion〉
    〈Motion type="walk" length="normal" repeat="3"〉〈/Motion〉
    〈Motion type="jumpup" height="low" repeat="1"〉〈/Motion〉
    〈Motion type="jumpdown" height="low" repeat="1"〉〈/Motion〉
    〈Motion type="run" length=short" repeat="2"〉〈/Motion〉
    〈Motion type="stand" repeat="1"〉〈/Motion〉
〈/MotionSequence〉
```

(a)

```
〈? xml version="1.0" encoding="UTF-8"?〉
〈! ELEMENT MotionSequence (Motion+)〉
〈! ELEMENT Motion (#PCDATA)〉
〈! ATTLIST Motion type
〈walk|run|stepup|stepdown|stepover|jumpup|jumpdown|jumpover|stand)'walk'〉
〈! ATTLIST Motion height 〈high|normal|low)'normal'〉
〈! ATTLIST Motion length (long|normal|short)'normal'〉
〈! ATTLIST Motion repeat CDATA'1'〉
```

(b)

Fig. 8.9 (a) Motion script for "locomotion" dataset; (b) XML DTD file for "locomotion" dataset

- "jumpup" to a high place with low height;
- "jumpdown" from a high place with low height;
- Then follows 2 cycles of "run" movement with short step-length;
- Finally ends with a "stand" motion.

If the users want to extend the motion script to adapt more kinds of motion elements and motion datasets, what they should do is to modify or replace the existing DTD file according to the content of motion dataset. For example, we can make a new DTD file and enable it to support the "fighting" motion elements in motion script (see Fig. 8. 10).

```
|<? xml version="1.0" encoding="UTF-8"?>
<! ELEMENT MotionSequences (Motion+))
<! ELEMENT Motion EMPTY>
<! ATTLIST Motion type
(hookpunch|straightpunch|roundhousepunch|uppercutpunch|frontkick|sidekick|
turningkick|sidestep|turningstep|backwardstep|forwardstep|overstep|ready-
pose)'readypose'>
<! ATTLIST Motion position (f|b)'f'>
```

Fig. 8. 10 A DTD file supporting "fighting" motion elements

8. 2. 3 Motion Generation

Before motion synthesizing, the motion scripts are analyzed by the standard XML parser which generates a group of 〈Motion〉 tags. Each 〈Motion〉 tag represents an independent motion. Then the corresponding motion clips are retrieved from database and stitched together to synthesize the final movement sequence.

8.2.3.1 Motion State Machine

Motion State Machine (MSM) is a connected graph. Each node in MSM represents a motion element from motion dataset. Fig. 8. 11 gives a MSM corresponding to the "locomotion" dataset. The directed links between nodes indicate state transitions, which mean that a state (motion element) can be followed by another state. Obviously, MSM is used to help us to determine proper transition between different motion states (motion elements) and work out a sequence of proper motions.

8.2.3.2 Motion Stitch

For each 〈Motion〉 tag, the corresponding motion clips can be retrieved from database by attribute values, which should be stitched together to get a complete motion sequence (see Fig. 8. 12).

In order to stitch two motion clips together, the well-studied motion transition technology should be applied. But as shown in Fig. 8. 11, in practice some motion ***A*** cannot be directly connected to motion ***B***. For example, there are two motion clips, the former is "walk" and the latter is

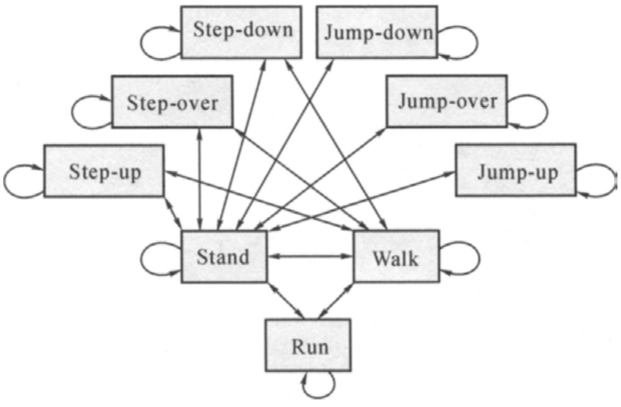


Fig. 8. 11 Motion state machine for "locomotion" dataset

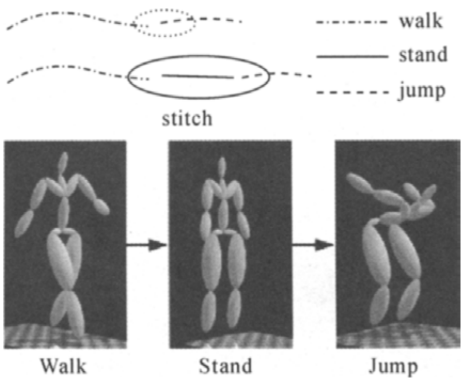


Fig. 8. 12 Motion Stitch. The dash-dot line represents a "walk" motion. It is followed by a "jumpup" motion, drawn in the green dot line. The padding part is a "stand" motion that is drawn in blue solid line. The motion clips should be stitched together to get smooth motion sequence

"jumpup" (see Fig. 8. 12). By the definition of MSM, there is no direct connection between "walk" and "jumpup" motion state, so a short "stand" motion clip should be inserted to make proper motion transition between them.

Because the motion clips from motion database are well segmented, here we only use the simplified method [16] to deal with motion transitions. A transition is a mapping of similar segments between two motions $\boldsymbol{A}$ and $\boldsymbol{B}$. The duration of the transition T_{trans} is calculated by taking the average of the length of the two blending regions:

$$T_{\text{trans}} = [(t_e^A - t_s^A) + (t_e^B - t_s^B))/2 \tag{8-5}$$

where t_s^A, t_e^A, t_s^B and t_e^B are the starting and the ending time of the transition region in each of the two motions.

Motion ***A*** and ***B*** are simply blended by fading out one while fading in the other. A monotonically decreasing blending function with a range and domain of [0,1] determines the relative contribution of the two motions. A sigmoid-like function, $\alpha = 0.5\cos(\beta\pi) + 0.5$, is used here. Over the transition duration, β moves linearly from 0 to 1 representing the fraction of the way through the transition intervals in each motion. If more natural transitions between motions are desired, approaches proposed in [11,17] can be used here.

8.2.4 Results and Discussions

In experiments, we set up a motion database and developed two classical demo applications based on our proposed motion script engine. For "locomotion" dataset, we developed an application to generate some navigation movement sequences offline. For "fighting" dataset, we developed a real-time application based on keyboard input to control the fighting actions of character.

8.2.4.1 Motion Datasets

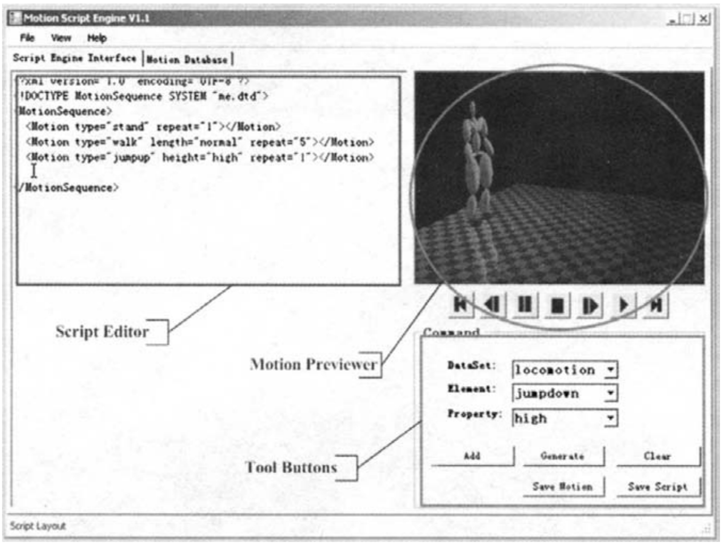
We set up a motion database which contains two datasets: "locomotion" dataset and "fighting" dataset. All of the motion clips are performed by real actors and captured at 60 Hz frame rate by an optical motion capture system from MotionAnalysis. The motion clips included in "locomotion" dataset are shown in Table 8.2. And the motion clips of "fighting" dataset are given in Table 8.3. The "fore" attribute value means that this action is performed by the arm or leg whose positions in front of the body. There are totally 19 actions in the "fighting" dataset. All of these motion clips are stored in a motion database created by Microsoft Access 2003.

Table 8.3 Motion clips in "fighting" dataset

Motion Element Name	Position		Description
	fore	back	
HookPunch	1	1	Hook punch
StraightPunch	1	1	Straight punch
RoundhousePunch	1	1	Roundhouse punch
UppercutPunch	1	1	Uppercut punch
FrontKick	1	1	Kick front
SideKick	1	1	Kick side
TurningKick	1	1	Kick with body turn
SideStep	1	1	Step side
TurningStep	1	1	Change the face direction
BackwardStep	1		Jump backward
ForwardStep	1		Jump forward
OverStep	1		Step forward
ReadyPose	1		Ready pose for fighting

8.2.4.2 An Offline Application for Navigation Movements Synthesis

In this experiment, we developed a demo application by C# in Microsoft Visual Studio 2005. This application is based on the "locomotion" dataset and the corresponding XML DTD file is defined in Fig. 8. 9(b). Users can write motion script and generate final motion sequence efficiently with the convenient graphical user interface provided by this application. As shown in Fig. 8. 13, users can use "Tool Buttons" to generate or edit the script in "Script Editor" directly. And the "Motion Previewer" is used to preview the generated motion sequence.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MotionSequence SYSTEM "me.dtd">
<MotionSequence>
  <Motion type="stand" repeat="1"></Motion>
  <Motion type="walk" length="normal" repeat="5"></Motion>
  <Motion type="jumpup" height="high" repeat="1"></Motion>

</MotionSequence>
```

Fig. 8. 13 Graphical user interface of offline application

Fig. 8. 14(a) gives the final motion sequence generated by our demo application with the scripts defined in Fig. 8. 9(a). As we can see, the engine inserted three "stand" motion clips into the entire motion sequence to ensure the proper movement transitions. As shown in Fig. 8. 14(b), the synthesized movement sequence can be applied into a simple scene to generate character animation sequence.

8.2.4.3 A Real-time Application for Fighting Character Controlling

Besides generating motion sequence with predefined scripts, our motion script engine can also be used in some real-time applications. For example, we can use our motion script engine to synthesize actions in real-time according to the online input of users. In this experiment, we developed a simple demo application by which users can control the character's fighting actions with keyboard input.

In this demo application, the "fighting" dataset listed in Table 8. 3 is

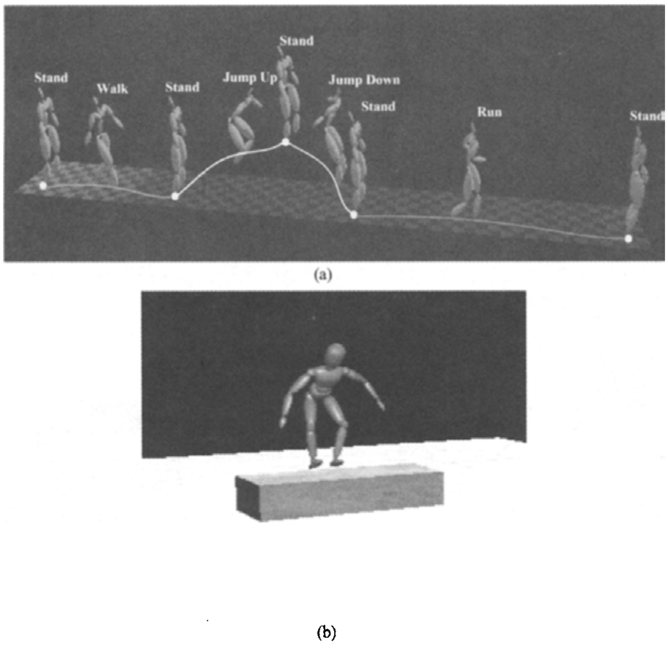


Fig. 8. 14 (a) Synthesized motion sequence; (b) Applying the resulting movement sequence into a simple scene

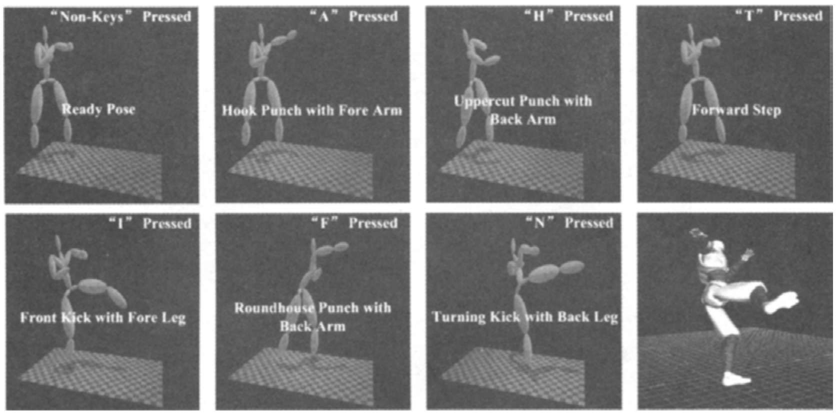


Fig. 8. 15 Synthesized "fighting" action sequence by keyboard input and the motion retargeting result on a character model

used and the corresponding XML DTD file is defined in Fig. 8. 10. In order to control the character's actions by keyboard, a group of hotkeys are defined to trigger the corresponding motion elements in "fighting" dataset, just like in the fighting games. Fig. 8. 15 shows the synthesized "fighting" action sequence and the motion retargeting result on a character model.

8.2.4.4 Discussions

In this section, we propose a flexible framework for human movement generation. Based on motion script engine and MoCap database, users can synthesize realistic human movement sequence efficiently by writing simple scripts or inputting discrete commands. In order to synthesize reasonable and smooth movement sequence, we integrate the Motion State Machine into the motion script engine which ensures the proper transitions between different motion elements intelligently. Furthermore, the definition of motion script is under the XML schema which guarantees the flexibility. Users can modify and edit the existing motion sequence easily and efficiently. And if the users want to make a movement sequence including some new motion elements, what they should do is only to supply the new motion clips and modify or replace the definition of the corresponding XML DTD file. Finally, we have developed two demo applications to show that our motion script engine works well and can be used as an embedded engine of motion synthesis into multiple applications, such as Virtual Reality, video game, animation software and sports simulation system, etc.

In the future work, we would like to improve the current work on three aspects. First, add some finer controlling functions into current motion script framework. By the current approach, users cannot control the details of character's movement by scripts, such as movement path controlling, motion style controlling, etc. So in the next research, we'll add motion path controlling and data based motion synthesis mechanism into the current framework, which might help users to generate more complex movement sequence. Second, extend the current motion script engine to support crowd movement synthesis, which is very useful for crowd animation and simulations. Third, extend motion script engine to support interactive human movements, such as handshaking and fighting between characters.

8.3 Automatic Generation of Human Animation Based on Motion Programming

As introduced at the beginning of this chapter, many methods focused on editing and synthesizing new motions have been proposed, which satisfy users' definitions using existing data. However, these methods cannot deal with the characters' interactions with virtual environment or automatically generate human-like character animations in virtual environment.

Interactive motion oriented methods are recently proposed to solve the problem. They focus on how to synthesize interactive motions in virtual environment or adapt existing motions derived from specific motion oriented methods to virtual environment, such as synthesizing interactive motions for manipulating some virtual objects, adapting existing motions to tasks of virtual environment navigation and producing navigation animations automatically, etc.

Yamane, et al. [18] presented a technique for synthesizing motions for object manipulation tasks in virtual environment. Given user-specified start and goal positions for an object, their algorithm relies on a randomized algorithm to find a feasible path for the manipulated object. The planning process is informed by a database of natural human postures for similar tasks and model-based balance and collision constraints, which attempts to combine the good features of model-based and data-driven approaches.

Choi, et al. [19] brought in probabilistic model to build up a graph of footprints in a scene. The result of their research showed that the character can respond to the virtual scene through the calculated footprint positions and orientations. Several methods to promote the result route were implemented in this paper.

In [20], a novel approach was proposed to evaluate a motion graph for navigation tasks in virtual environment. They defined metrics for evaluating expected path quality and coverage of specific motion graph for a given environment. An improved motion graph was further embedded into a particular environment to get more natural and feasible navigation animations.

In digital entertainment applications, especially in video games and animation films, there are so many interactions between characters and virtual environment. Given a virtual environment, animators must adapt the characters' movements to it manually, which is a boring and time-consuming job. We are researching how to help animators to build up sequences of realistic motions interactively or automatically. The research in [19] is something near the point. But actually they didn't handle the overlapping surfaces. And they did the motion mapping only within tens of clips and the users cannot change or refine the results interactively.

8.3.1 Overview

8.3.1.1 Clip Families

Generally speaking, there are many kinds of motions that can be performed by characters in virtual environment, such as walking, running and jumping, etc. In this section, we will show a ten-family-structure motion scripting in the following parts. We define a motion clip as:

$$\boldsymbol{m}=\{\boldsymbol{f},\boldsymbol{P}\} \tag{8-6}$$

where $\boldsymbol{m}$ is a motion clip, $\boldsymbol{f}$ is the family it belongs to, and $\boldsymbol{P}$ is the parameters vector corresponding to the specific family. A family is a set of motions of the same type, which has a few parameters (see Table 8.4). Clip

Table 8.4 Parameters and bias coefficients of motion family

Motion Family	Parameters	Bias
Walk	path length	0.1
Run	path length	0.2
Step down	height, width	0.3
Jump down	height, width	0.4
Step up	height, width	0.5
Jump up	height, width	0.6
Step over	height, width	0.7
Jump over	height, width	0.9
Leap over	height, width	0.8
Stand	none	0.0

families and corresponding parameters will be used in motion selection and motion acquisition processing.

8.3.1.2 Motion State Machine (MSM)

In Fig. 8.16, the directed links between states indicate state transitions, which mean that a state (a motion from a family) can be followed by another state (a motion from another family, or another motion from the same family). What we do to generate a sequence of animation is to program on this MSM, work out a sequence of proper motions and stitch them together.

As shown in Fig. 8.16, our system is based on the motion capture techniques. The MSM plays a role as the categorizing standard and gives us instructions to collect motion clips and build the database. All motion clips are classified and parameterized after being captured and are stored in the

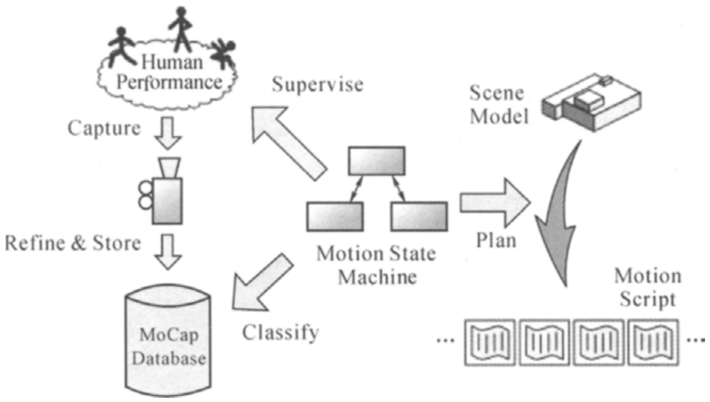


Fig. 8.16 The role of motion state machine. We build the database by capturing real human performance. Then given a scene model, the system is able to generate motion scripts for all planning tasks. And these operations are all based on the motion state machine

database which can be indexed by their family info and parameters. MSM also helps us to determine proper transition between motion clips belonging to different families.

8.3.1.3 System Architecture

Our system consists of five parts: MoCap database, motion acquisition module, motion planning module, visual user interface module and animation generation module (see Fig. 8. 17).

Firstly, MoCap data are captured, parameterized, stored and indexed in MoCap database. Users can use visual UI module to change or refine motion scripts returned by motion planning module. Then motion acquisition module use motion scripts to retrieve proper motion clips from MoCap database, and these original clips are used directly or edited according to constraints given in motion scripts. Finally these motion clips are delivered to animation generation module, and the result navigation animations are generated automatically and returned to users.

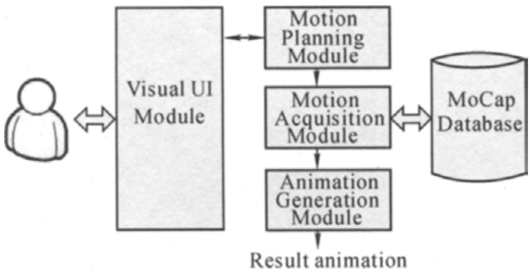


Fig. 8. 17 System architecture

8.3.2 Roadmap Generation

In a 3D scene, we use the concept of roadmap to build up a planner for the path [19,21], and other researches have already brought out some implementations using roadmap. Here we implement a structure called pleat, i. e. a group of surfaces, to handle mesh information of virtual environment.

8.3.2.1 Pleat Building and Node Sampling

We first group the surfaces in the scene mesh which are continuous and have a low slope. We simply join two surfaces together if two vertices of each surface are very close and their normals are leaner than a threshold. After parsing all the surfaces in the scene mesh, we have several uneven surface groups which are called pleats. Each pleat, consisting of several surfaces, can hold the character on itself and let the characters move around without jumping or stepping (see Fig. 8. 18).

The reason why we build pleats is to reduce calculation in the roadmap node sampling, and to give bounds to the user interacting operations. This

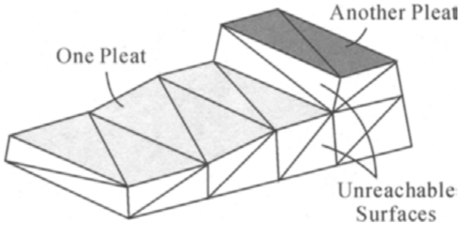


Fig. 8. 18 The pleats: there are two pleats in this figure. One is shaded in light grey and the other is in dark grey, other faces without shaded are unreachable

will be discussed in the later section.

We randomly sample the roadmap nodes by a probability model. Firstly we generate nodes by uniformly randomizing the positions. The density of the nodes is determined by the average translation value of common human motions, such as the average length of a single walking step. In consideration of efficiency, each pleat is treated as its projection on xoz plane (assuming that the direction of gravity is negative to the y axis). Because invalid faces are already excluded by forming the pleats, there is no need to do the verification.

8.3.2.2 Node Connection and Enhancement

Usually there may be a huge number of nodes in a virtual scene. We connect each node to its K-closest neighbors in order to save time. As a result the calculation of the path finding algorithm is limited to $O(NK+N\log N)$ (Dijkstra algorithm, N is the number of nodes). We also give the connection a threshold on its length to prevent the character from reaching a distant place in a single action. This is a preset value related to the average length of a human action, such as the length of a single walking step.

After generating nodes and edges, we then apply an enhancement to the roadmap with the method mentioned in [19]. In general, the distribution of sampling nodes is uniform which will decay in a narrow area or a difficult region, for example, a valley between unreachable cliffs. So the probability model is employed to adjust the density of nodes and the interconnections in this area. For each node v in set V of all nodes in the roadmap, the probability density function can be described as:

$$P(v \mid V) = \frac{1}{i_v + 1} \Big/ \sum_{u \in V} \frac{1}{i_u + 1} \tag{8-7}$$

where i_v is the number of edges connected to node v. This density function gives the difficult region more nodes to guarantee that the interconnection in this region will not decay any more (see Fig. 8. 19(a)). Empirically the number of additional nodes between 1/3 and 1/2 of the initial number yields good performance [22].

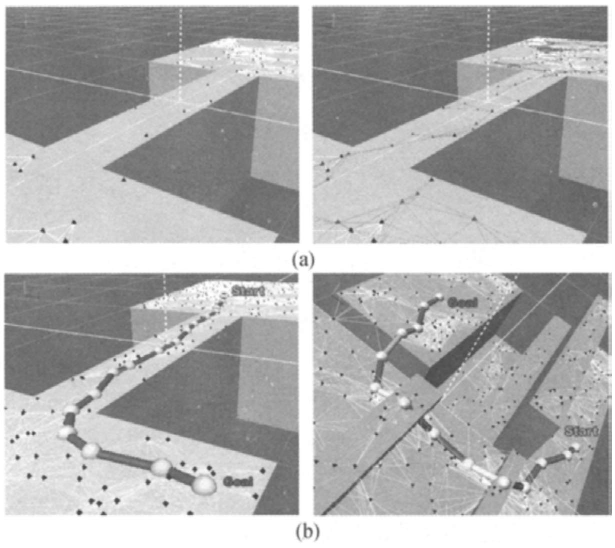


Fig. 8. 19 (a) The left figure shows the original roadmap without enhancement. On the bridge the interconnections decay for the lack of nodes. The right figure shows the enhanced roadmap and this problem is fixed. (b)Route planning result. Here shows the result route of an example scene. The spheres represent the node of the route and the sticks refer to motion families. The small dots are nodes of the roadmap. And the filaments are connections of the roadmap

8.3.3 Route Planning

After the generation of the roadmap, this stage is going to provide a sequence of motions that connect the start position and goal position. We call this sequence a route. It contains not only the geometric information of the path, but also the motions for each section of the path. The main idea is that our system chooses a motion family to apply to this connection in roadmap graph and give it a weight. Then we do a path finding procession on the weighted roadmap to get an optimal route.

8.3.3.1 Motion Selection

For every connection in roadmap, it means a character can move along from one terminal to the other. Normally we assume all the motions are walking straight, (e. g. walking on a large plane). However, some geometric constraints affect the connection. For example, connections across ditches or blocks do not allow a simple walking motion to apply and the stepping motion should be brought in.

Obviously each connection costs the character some efforts to travel along. In our system, the cost consists of two parts: the distance and the human bias, showing as follows:

$$c = \alpha w_d + \beta w_b \tag{8-8}$$

where w_d is the distance between the two terminals and w_b is the bias coefficient. The constant coefficients, α and β, normalize the weights.

Distance is the most intuitive cost while planning. And for the bias coefficient, Table 8. 4 weighting motions of various families shows how humans bias different motions (Notice that the "stand" motion has an infinitive coefficient since we don't want to add recursive "stand" loops in any cases). This table shows the basic configurations for different motion families. Users can define their own values for this table to achieve their desired motion selection results.

Besides the bias coefficients, we use some thresholds for "step" and "jump" relative motion selection. If the height of hurdle is higher than half a meter, or the width of hurdle is wider than a meter, "jump" should be used to travel through it instead of "step". It's uncomfortable for a human to travel through higher or wider hurdle with "step" motions. If the width of hurdle is wider than 2 meters, "leapover" motion should be used to travel through it. These thresholds can be set by users or be derived from all usable "step" or "jump" relative motion clips in MoCap database.

8.3.3.2 Path Finding

With the cost for each connection, we can perform a path finding algorithm to tell the best route. When the start point and the goal point are set by animators, we simply treat them as new nodes. We then connect them to the surrounding nodes on the roadmap and give cost values to the corresponding connections. Finally the path finding algorithm is performed by use of Dijkstra algorithm (Fig. 8. 19(b) gives the final route planning results).

8.3.4 Interaction and Optimization

8.3.4.1 Path Refining

Since nodes are sampled randomly and the modifications from animators may be unbending, the result route may have some flaws such as the zigzag pattern (see Fig. 8. 20). To make the path geometrically smoother, we consider the node positions of the route as discrete signals and apply a filter mask to them[23]. A simple mask can be defined as:

$$P_i' = Filter(P_i) = \frac{1}{16}(P_{i-2} + 4P_{i-1} + 6P_i + 4P_{i+1} + P_{i+2}) \tag{8-9}$$

where P_i is the original position of the ith node on the route, $P_i{}'$ is the modified position of the ith node on the route.

8.3.4.2 Immediate Interaction

Animators would not easily work out the scripts manually in a complex

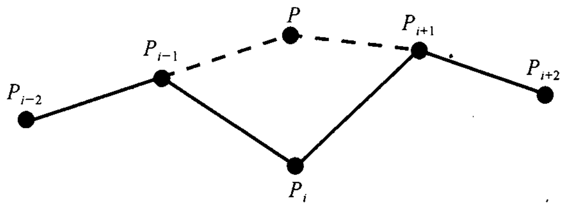


Fig. 8. 20 Path refining

virtual scene with mazes or obstructions. We need an automatic planner to work out the path and motion scripts on a maze like terrain or a scene with some stairs, blocks, ditches or something else. Although the outcome scripts can be somewhat optimized, usually they will not be fully accepted by animators. The interactive modification is thus supported in our system.

Through the previous stages, we can get the route between any start-goal pairs. We then push the joy stick to animators. They may do some modifications to the route, including changing the positions of the nodes or motion selection results (see Fig. 8. 21(a)).

Here the structure of pleat is employed again to limit the position chan-

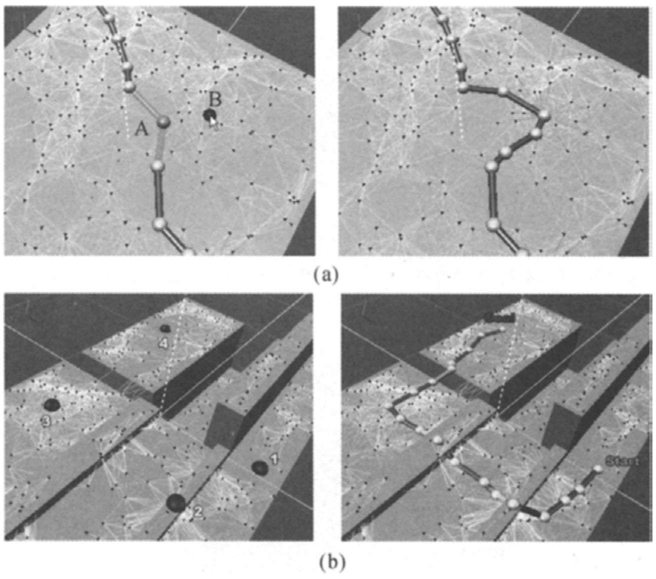


Fig. 8. 21 (a) Left is the planning result given by system. Users can drag sphere A to the position of sphere B by mouse. Right is the final route after modifications; (b) The left figure shows the points given by users. Final route and corresponding motion selection results given by system are showed in they right figure

ging of each node. If a node from a calculated route is moved to another pleat, the possibility of how to plan the new route will explode vastly, because not only continuous motion, but also overleap-like motion such as jumping and stepping should be considered. Therefore, the node position modifications should be limited in its initial pleat.

Animators can also give several points telling the system to follow instead of only the start point and the goal point and run the route planning. The sequence of points given by animators is treated as a list of routes and the route planning is applied to each segment (see Fig. 8. 21(b)). In our system, changes can be applied in real-time.

8.3.4.3 Motion Refining

Motion refining is very important for getting reasonable motion selection results. All of the motion selection results embedded in the route should be validated. But unfortunately they may be unreasonable, especially in joint region of different pleats. For example, in a route planning result, the character should step up from P_i to P_{i-1} and then step down from P_i to P_{i+1}. But if the distance between P_{i-1} and P_{i+1} is small, these two motion should be substituted by a stepping over motion between P_{i-1} and P_{i+1} (see Fig. 8. 22). The same refining processing should be applied to "jump up" and "jump down" motions when the width of hurdle is small.

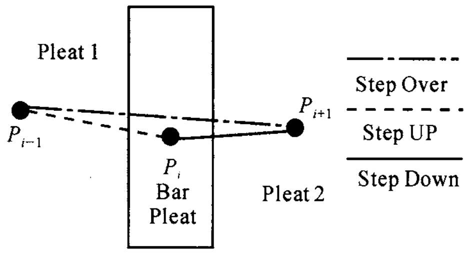


Fig. 8. 22 Motion refining

8.3.4.4 Scripts Generation

The neighboring connections on the best route with the same motion family are grouped to generate motion scripts. Since we want animators to access and modify the result freely, we use the XML pattern to store the motion script. Fig. 8. 23 is an example of motion scripts.

8.3.5 Motion Acquisition

In motion acquisition module, motion clips similar to the movements described in the motion scripts are retrieved and some constraint-based methods are applied on them to get final desired motions.

8.3.5.1 Motion Retrieval

Here a simple motion capture system is used to record real human performances. When motion clips from MoCap system are added into MoCap

```
<? xml version="1.0" encoding"UTE-B"?>
<! DOCTYPE MotionSequences SYSTEM"ms.dtd">
<MotionSequences>
<Motion type="walk">
  <path>
    <point>X1,Y1,Z1</point>
    <point>X2,Y2,Z2</point>
          ...
    <point>Xn,Yn,Zn</point>
  </path>
</Motion>
    ...
<Motion type="stepover">
  <path>
    <point>XX1,YY1,ZZ1</point>
    <point>XX2,YY2,ZZ2</point>
    <point>XX3,YY3,ZZ3</point>
<point>XX3,YY3,ZZ3</point>
  </path>
</Motion>
</MotionSequences>
```

(a)

```
<? xml version="1.0" encoding"UTE-8"?>
<! ELEMENT MotionSequences(Motion+)>
<! ELEMENT Motion (Path)>
<! ATTLIST Motion type (walk|run|leapover|stepup|stepdown|stepover|jumpup|
   jumpdown|jumpover)>
<! ELEMENT Path(point,point+)>
<! ELEMENT point( #PCDATA)>
```

(b)

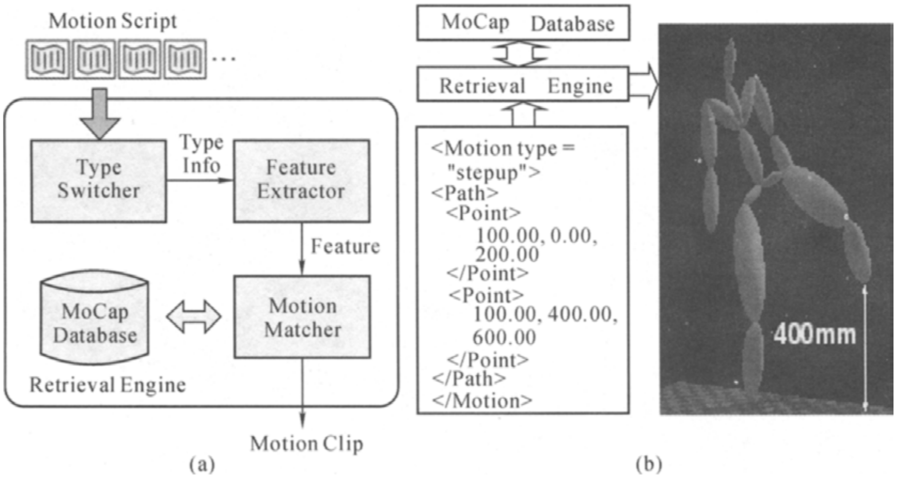
Fig. 8.23 (a) Motion scripts; (b) Corresponding DTD file

database, we annotate their motion family info manually and the parameters corresponding to different motion families are extracted and stored automatically (see Table 8.2). In MoCap database, the family info and corresponding parameters are used to index each motion clip.

Motion scripts describe the characters' movements in virtual environment. So in order to produce human-like character animation, raw motion clips which are similar to the movements described in motion scripts should be retrieved from MoCap database first. Here we implement a simple engine to retrieve motion clips from MoCap database by motion scripts.

As shown in Fig. 8.24(a), type switcher first judges the motion family info, and then asks feature extractor to extract corresponding parameters from movements which are described in motion scripts. Finally motion matcher uses these parameters to find optimal motion clips from MoCap database and return them.

All of the parameters used to query are extracted from the 〈Point〉 set



```
<Motion type =
  "stepup">
<Path>
  <Point>
     100.00, 0.00,
     200.00
  </Point>
  <Point>
     100.00, 400.00,
     600.00
  </Point>
</Path>
</Motion>
```

Fig. 8. 24 (a) A simple retrieval engine. The input scripts come from the result of motion planning module; (b) A motion retrieval example. The left bottom is the motion scripts segment corresponding to a "step over" motion. The right is the motion clip returned by motion retrieval engine. The unit in motion scripts is millimeter (mm)

of the motion scripts. For "walk" and "run" motion, all of the ⟨Point⟩s in ⟨Path⟩ segment are used to calculate the "path length" parameter (all of the points are projected onto xoz plane for calculating). For remainder motion, the first and the last ⟨Point⟩ in ⟨Path⟩ segment are used to calculate the "width" parameter (the distance on xoz plane between these two points) and the difference between the maximum and the minimum of all ⟨Point⟩s' y axis value is set to "height" parameter. A motion retrieval example is given in Fig. 8. 24(b).

8.3.5.2 Parameterized Motion Synthesis

For retrieval results which do not exactly match the parameters given in motion scripts, some constraint-based motion editing and synthesis methods are applied. Parameterized motion synthesis method is used to get "step" (up, down and over), "jump" (up, down and over) and "leap over" motions with desired "height" and "width" parameters. For example, a "step up" motion with height of 0. 5 m is desired but in database only "step up" motion clips with height of 0. 4 m and 0. 6 m are usable.

The approach proposed in [24] is used to parametrically synthesize desired motions. Given a group of similar motion clips which belong to the same motion family, we automatically register them and apply blending techniques to create a continuous space of motions. Finally we can use parameters given in motion scripts to synthesize desired motion in this space of motions (see Fig. 8. 25). Detailed description of this approach can be found in [24].

8.3.5.3 Motion Path Editing

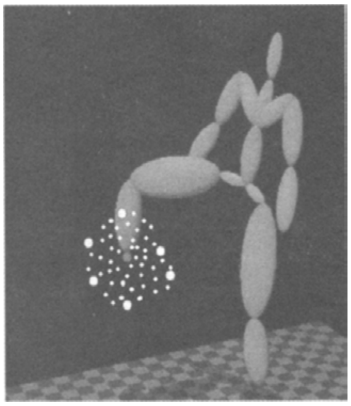


Fig. 8. 25 Visualization of the target locations for the right foot of "step up". Large yellow dots show parameters of retrieval result motions; large red dot shows parameter of the synthesized motion; small grey dots are sampled parameters of space of motions

Given a motion clip from database, how can we adapt it to the new path described in motion scripts (see Fig. 8. 26(a))? Simply changing the length and shape of motion path will result in footskating and orientation error of the character. With the approach proposed in [25], we can retarget original motion path to new path efficiently.

Firstly, we stretch the path length of original motion to the path length given in motion scripts and additional frames are added so that the motion arrives at the end of the path by repeating the cycle motions. Then path $\boldsymbol{P}$ of stretched motion and path $\boldsymbol{P}'$ given in motion scripts are parametrically represented by two B-splines. We just use parameters of $\boldsymbol{P}$ to re-parameterize the path $\boldsymbol{P}'$, which reserves the time and space constraints of original motion in new motion. Finally local frame coordinates of original motion are used to adapt the orientation of character in new motion. Detailed method is well described in [25]. Motion path retargeting result is showed in Fig. 8. 26(b).

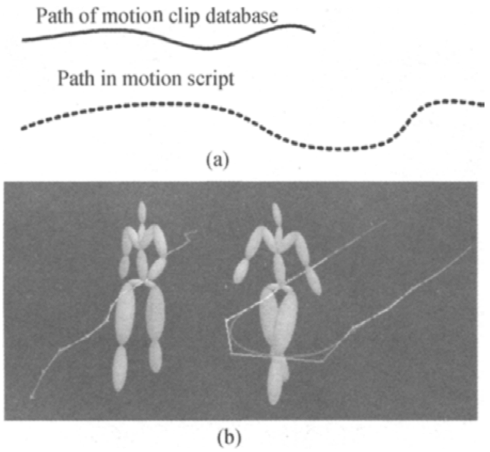


Fig. 8. 26 (a) Path editing. The solid curve represents the path of a motion captured and stored in the database. The dot curve represents the path in the outcome route; (b) Motion path editing result. The left is an original motion. The right is the motion with new path whose length and shape are changed

8.3.6 Animation Generation

The animation generation method used here is the same as that of Sect. 8.2.3.

8.3.7 Results and Discussions

In experiments, we captured about 300 motion clips with varied parameters which were divided into 9 types (see Table 8.4). And another "stand" motion clip was used to motion stitch.

In order to confirm the feasibility of our approach, we implement a prototype system and get some encouraging results. As shown in Fig. 8.27 (a), given a complex virtual scene with start and goal points, the motion

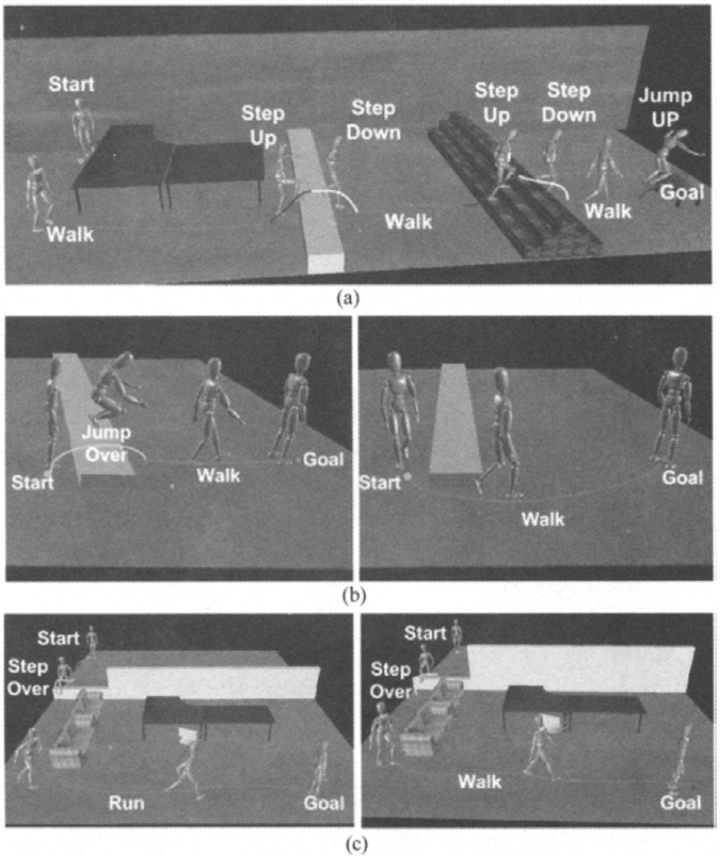


Fig. 8.27 (a) Animation result in a complex virtual environment; (b) Result with basic configuration of motion bias coefficients (right). The motion bias coefficient of "jump over" motion is reset so that users can get new animation result (left); (c) Result generated by system (right). User can change motion selection result interactively (using "run" motion instead of "walk" motion) and get new animation result (left)

programming results and animation sequences are generated automatically. Users can also change the basic configuration of motion bias coefficients to achieve different motion programming results and animation sequences. In Fig. 8. 27(b), bias coefficient of "jump over" motion is set with a small value, so different animation sequences are generated. In order to get varied animation results in the same virtual scene, users can change the motion selection results interactively. As shown in Fig. 8. 27(c), users can use "run" motion instead of "walk" motion to get a new animation sequences.

The framework of our system is developed with Java 3D and works on a PC with PentiumIV 2. 8 GHz CPU and 1 G memory. The functions of motion retrieval, motion path editing, parameterized motion synthesis and motion stitch are developed by C++ and wrapped as DLL libraries which can be called in our system. And these DLL libraries can be conveniently replaced by the newer or better ones. Some performances of our system are given in Table 8. 5.

Table 8. 5 Performances of our system

Motion clip parameters extraction	0. 031 s/clip (average time)	
Motion planning (tiny scene, 20 m×10 m, about 600 nodes and 4,000 lines)	Roadmap generation	Route planning
	2. 07 s	0. 34 s
Motion planning (large scene, 70 m × 60 m, about 15, 000 nodes and 167,000 lines)	Roadmap generation	Route planning
	13. 9 s	34. 1 s
Motion scripts parsing and clip acquisition	0. 628 s/clip (average time)	
Motion stitch	0. 037 s/2 clips (average time)	

In our framework proposed above, system can automatically program human motions and generate corresponding animation sequences in a virtual environment with specified start and goal points, and varied animation sequences can be achieved by two methods: (1) change the basic configuration of motion bias coefficients; (2) change the motion selection results interactively by the visual user interface of our system.

In addition, many novel specific motion oriented methods can be used in our framework to get more realistic animation results, such as motion transition method to get more natural transitions between motion clips, motion synthesis method to get desired motions, motion retrieval method to get more precise retrieval results, etc. All of these methods can be implemented as a component and added into our framework conveniently.

For a virtual scene with specified start and goal points, final motion scripts can also be stored as standard XML files and reused to modify or generate animations efficiently without repeating path planning and motion selection.

Currently this framework is only for motion programming of a single character in a known virtual scene. In future we hope to extend it for dynamic 3D scene and multiple characters, which is more useful for animation productions, games and motion simulations.

References

1. Ko H, Badle NI. Straight line walking animation based on kinematic generalization that preserves the original characteristics. In: GI'93, pp. 9-16, Springer-Verlag, 1993.
2. Unuma M, Anjyo K, Takeuchi R. Fourier principles for emotion-based human figure animation. In: SIGGRAPH '95, pp. 91-96, ACM Press, 1995.
3. Wiley DJ, Hahn JK. Interpolation synthesis for articulated figure motion. In: VRAIS'97, pp. 157-160, IEEE Computer Society, 1997.
4. Witkin A, Popvic Z. Motion warping. In: SIGGRAPH'95, pp. 105-108, ACM Press, 1995.
5. Bruderlin A, Williams L. Motion signal processing. In: SIGGRAPH'95, pp. 97-104, ACM Press, 1995.
6. Cohen MF. Interactive space time control of animation. In: SIGGRAPH'92, pp. 293-302, ACM Press, 1992.
7. Liu Z, Gortler SG, Cohen MF. Hierarchical space time control. In: SIGGRAPH'94, pp. 35-42, ACM Press, 1994.
8. Gleicher M. Motion editing with space time constraints. In: Symposium on Interactive 3D Graphics, pp. 139-148, ACM Press, 1997.
9. Gleicher M, Litwinowicz P. Constraint-based motion adaptation. Journal of Visualization and Computer Animation, 9(2): 65-94, 1998.
10. Popovic Z, Witkin A. Physically based motion transformation. In: SIGGRAPH'99, pp. 8-13, ACM Press, 1999.
11. Rose C, Guenter B, Bodenheimer B, Cohen MF. Efficient generation of motion transitions using space time constraints. In: SIGGRAPH'96, pp. 147-154, ACM Press, 1996.
12. Lee J, Shin SY. A hierarchical approach to interactive motion editing for human-like figures. In: SIGGRAPH'99, pp. 39-48, ACM Press, 1999.
13. Kalisiak M, van de Panne M. A grasp-based motion planning algorithm for character animation. http:// www. dgp. toronto. edu/_mac/masters/, 2002-01.
14. Fua P, Gruen A, Plankers R, D'Apuzzo N, Thalmann D. Human body modeling and motion analysis from video sequences. In: ISPRS'98, pp. 866-873, 1998.

15. Liu X, Zhuang Y, Pan Y. Video based human animation technique. In: Multimedia'99, pp. 353-362, ACM Press, 1999.
16. Rose C, Cohen MF, Bodenheimer B. Verbs and adverbs: multidimensional motion interpolation. IEEE Computer Graphics and Applications, 18(5): 32-40, 1998.
17. Kovar L, Gleicher M. Flexible automatic motion blending with registration curves. In: SIGGRAPH'03, pp. 214-224, ACM Press, 2003.
18. Yamane K, Kuffner J, Hodgins JK. Synthesizing animations of human manipulation tasks. In: SIGGRAPH'04, pp. 532-539, ACM Press, 2004.
19. Choi MG, Lee L, Shin SY. Planning biped locomotion using motion capture data and probabilistic roadmaps. ACM Transactions on Graphics, 22(2): 182-203, 2003.
20. Reitsma PSA, Pollard NS. Evaluating motion graphs for character navigation. In: SIGGRAPH/EuroGraphics Symposium on Computer Animation, pp. 88-98, ACM Press, 2004.
21. Salomon B, Garber M, Lin MC, Manocha D. Interactive navigation in complex environments using path planning. In: Symposium on Interactive 3D Graphics, pp. 41-50, ACM Press, 2003.
22. Kavraki L, Kolountzakis M, Latombe JC. Analysis of probabilistic roadmaps for path planning. In: International Conference on Robotics and Automation, pp. 3020-3025, IEEE Computer Society, 1996.
23. Lee J, Shin SY. General construction of time-domain filters for orientation data. IEEE Transactions on. Visualization and Computer Graphics, 8(2): 119-128, 2002.
24. Kovar L, Gleicher M. Automated extraction and parameterization of motions in large data sets. In: SIGGRAPH'04, pp. 558-568, ACM Press, 2004.
25. Gleicher M. Motion path editing. In: Symposium on Interactive 3D Graphics, pp. 195-202, ACM Press, 2001.

Index