COMBINED FILTERING AND KEYFRAME REDUCTION FOR MOTION CAPTURE DATA

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND THE INSTITUTE
OF ENGINEERING AND SCIENCE
OF BİLKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Onur Önder

July, 2007

ABSTRACT

COMBINED FILTERING AND KEYFRAME REDUCTION FOR MOTION CAPTURE DATA

Onur Önder
M.S. in Computer Engineering
Supervisors: Prof. Dr. Bülent Özgüç and
Assoc. Prof. Dr. Uğur Güdükbay
July, 2007

Two new methods for combined filtering and key-frame reduction of motion capture data are proposed. Filtering of motion capture data is necessary to eliminate any jitter introduced by a motion capture system. Although jitter removal is needed to obtain a more realistic animation, it may result in an over-smoothed motion data if it is not done properly. Key-frame reduction, on the other hand, allows animators to easily edit motion data by representing animation curves with a significantly smaller number of key frames. One of the proposed techniques achieves key frame reduction and jitter removal simultaneously by fitting a Hermite curve to motion capture data using dynamic programming. Another method is to use curve simplification algorithms on the motion capture data until the desired reduction is reached. In this research, the results of these algorithms are evaluated and compared. Both subjective and objective results are presented.

Keywords: Motion Capture, Keyframe Reduction, Curve Fitting, Curve Simplification, Noise Filtering.

ÖZET

BİRLEŞİK OLARAK HAREKET YAKALAMA VERİLERİNİN FİLTRELENMESİ VE ANA ÇERÇEVELERİN AZALTILMASI

Onur Önder
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticileri: Prof. Dr. Bülent Özgüç ve
Doç. Dr. Uğur Güdükbay
Temmuz, 2007

Birleşik olarak hareket yakalama verilerinin filtrelenmesi ve ana çerçevelerin azaltılması için iki yeni yöntem sunulmuştur. Hareket yakalama bilgilerinin süzülmesi, hareket içerisindeki gürültünün yok edilmesi için gereklidir. Hareket içerisindeki gürültünün yok edilmesi daha gerçekçi bir canlandırma sağlar. Ancak gerekli ölçüde yapılmadığı takdirde çok fazla düzleştirilmiş, gerçekçi olmayan bir canlandırma ortaya çıkarır. Diğer yandan, ana çerçevelerin azaltılması, bu verilerin canlandırma eğrileri ile tanımlandığı için, canlandırıcıların bu verileri çok daha az ana çerçeve sayısı ile kolaylıkla düzenleyebilmelerini sağlar. Belirtilen yöntemlerden ilki, dinamik programlama yöntemi ile beraber hareket yakalama verisine Hermite eğrisi oturtarak aynı anda hem ana çerçeve azaltımını yapmaktadır, hem de verilerin gürültüden ayrılmasını sağlamaktadır. Diğer bir yöntem ise aynı sonucu eğri basitleştirme yöntemlerini istenilen duyarlılık erişilinceye kadar kullanımıdır. Bu araştırmada bu yöntemlerin sonuçları değerlendirilmiş ve karşılaştırılmıştır. Ek olarak öznel ve nesnel sonuçlar da sunulmuştur.

Anahtar sözcükler: Hareket Yakalama, Ana Çerçevelerin Azaltılması, Eğri Oturtma, Eğri Sadeleştirme, Gürültünün Filtrelenmesi.

Acknowledgement

I would like to gratefully acknowledge the support and supervision of Prof. Dr. Bülent Özgüç and Assoc. Prof. Dr. Uğur Güdükbay who allowed me to have the opportunity of pursuing this research. I would like to thank for the opportunities that European Commission 6th Framework Network of Excellence project "Integrated Three-Dimensional Television - Capture, Transmission and Display" provided to pursue my research. I would like to express my gratitudes to Prof. Dr. Özgür Ulusoy, Prof. Dr. Enis Çetin and Assist. Prof. Dr. Tolga K. Çapın for their comments and suggestions. I also would like to thank Dr. Tanju Erdem for many stimulating discussions and his support on my research topic.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Animation techniques are rapidly changing through the evaluation of technology and computers. One such technique is simulating the real world in a virtual and confined environment. For such purposes some motion capture equipments are developed and in conjunction with the computers, the motion of a real world object (such as a human) can be replicated in this virtual world. Animators use this technique to create more real-world like animations since they are the near perfect duplications of the actual motion. In this chapter, some general information about how motion capture systems work and how the animators work with the motion capture data will be given, then the problem that this thesis will cover will be presented.

Motion capture saves a lot of time for the animators. Usually animations consists of motions from the real world and people want them to obey the laws of physics. If the desired animation has to obey the laws of physics and has to be done in a real environment, animators usually choose the motion capture techniques. The animated virtual object does not have to exist, it can be a chair walking or a tin talking like a man. Animators either use already captured motions such as walking or running of a human, then apply it into a virtually generated and animated character; or they request a motion capture studio to be constructed and the required actions to be captured. In this process, there are two key problems, animators should be able to control the action or re-use it multiple times easily, and the motion information has to be captured so that it is not effected by any means of noise and its data stored efficiently.

There are motion capture libraries that contains previously captured motions for some common actions. This enables an animator to use these pre-recorded data but presents another problem; what if the animator wants to alter some of the data? For instance, if the animator has a human motion capture data and he/she desires to make the captured body's arm to be a little further away from the body, he/she has to manually edit the captured data frame by frame. Since motion capture systems record the position of a predefined joint or a part throughout the time, the animators usually end up having data for each joint for each frame (which is the smallest unit of time for a change in an animation). If the animator wants to change the rotation data for the arm part of the animation, he/she has to modify each frame manually. There are tools that help this process, but even with the tools, animators have to make sure that the animation flow is continuous and does not disturb the viewer. Even if we have a 10 seconds of animation with 30 frames/second, the animator has to check 300 frames for one joint for its continuity and completeness. This number increases as the number of joints and the number of frames increase and make the problem more complex for an animator.

The second important thing in a motion capture system is to make sure that the data is not effected by noise and it is stored in an efficient way. The equipment used for the motion capture systems are usually based on transmitters placed on the joints of an actor (these can be as simple as light reflectors in certain colors or they can be as complicated as radio wave transmitters transmitting its identification number and possibly its orientation or position) and multiple receivers that are recording the transmitters' positions. After capture, the gathered information is processed and the positions or orientations of individual points in space are recorded. Since it is not a robust operation, the exact motion may not be recorded, but a motion very similar to its real counterpart is recorded. This brings in the question of noise. Some motion capture systems try to decrease the noise in the recorded data as much as possible but nearly always the captured motion is analyzed by an animator who filters the captured data and makes sure that the captured data represents the action that is needed for the animation. After the post processing of the motion captured data, it is usually recorded on a digital media. Since we have the data for the orientation or position of each and every joint, this may require some notable storage space for each joint for every frame. If the animation is long and the number of recorded joints are high, the captured data has to be stored in an efficient way. Of course standard compression algorithms would work on the situation but this may increase the operation time for accessing the data when needed.

Apart from these major problems in a motion capture system, a solution may lie in the techniques that the animators are already using. In the past, the cartoon animations were done in two phases as Mealing [15] and Izani et al. [10] describe. In the first phase, key points in the cartoon's plot are drawn (they are called key frames), and later in the second phase, people called inbetweeners would draw the frames that lie in between the two keyframes. Today the computer graphics animators usually work with the keyframes, and while rendering, the computers fill the gaps in between.

Keyframes allow animators to define a beginning and an end point for a joint's animation, and possibly some other parameters such as the curvature and continuity around the beginning and the end points; and every frame between these two frames are handled and calculated automatically. With the calculated in between frames, rendering can be done. However, while calculating the motion in the frames between the keyframes, degree of the curve is important, which also defines certain behaviors of the motion. The states of every joint (position and orientation) can be easily found when the keyframes are fully known, thus it allows to render the motion without much operation. On the other hand the motion capture data always consists of the states of each individual frame, but it lacks any possible keyframe data that can be used. If by any means, the motion capture data can be expressed in terms of keyframes, the states of the joints in the motion of the animation, can easily be altered and modified. The animators who will modify the motion would need to work on a small set of frames. For instance, if the animator would want to let the actor's hand in the motion to move in another path, the animator would only need to change the beginning

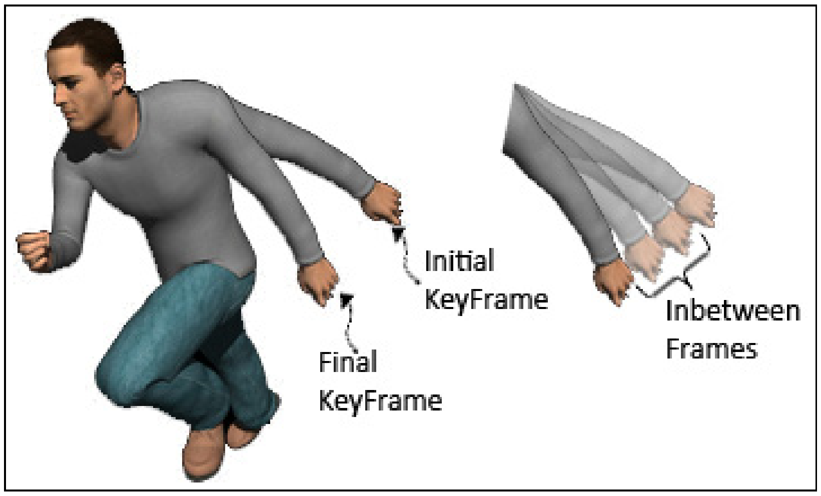


Figure 1.1: In between frames calculated between the initial and final keyframes.

and the ending points of the hand's new motion, which would be defined by the keyframes at the beginning and the end point of the motion. In Figure 1.1, the motion between the initial and final keyframes is illustrated.

If one can select some "good" keyframes over the motion path, the motion can be expressed with a smooth path, which would not only eliminate immediate noises but also enable the animator to work on the motion easily. The ability to choose a "good" keyframe would need some specifications if the desired outcome needs to eliminate the noise but also leave the details of the rapid actions in place (such as the hand movement of an old person who has trembling hands). Since one may either desire a motion with high number of rapid movements (which needs to have a high ratio of the number of keyframes over the number of ordinary frames), or desire a motion with more smooth and longer motions (which needs to have a low ratio of the number of keyframes over the number of ordinary frames); some parameters, with respect to the identification of keyframes, should be declared. By using these parameters, the detailed motions can be preserved as well as any possible noise can be eliminated. From another point of view, by using these keyframes, the overhead of saving the position of each and every joint in every frame can be reduced. As shown in Figure 1.2, by only saving the keyframe data, the required storage space is decreased by the amount of the ratio of keyframes over frames.

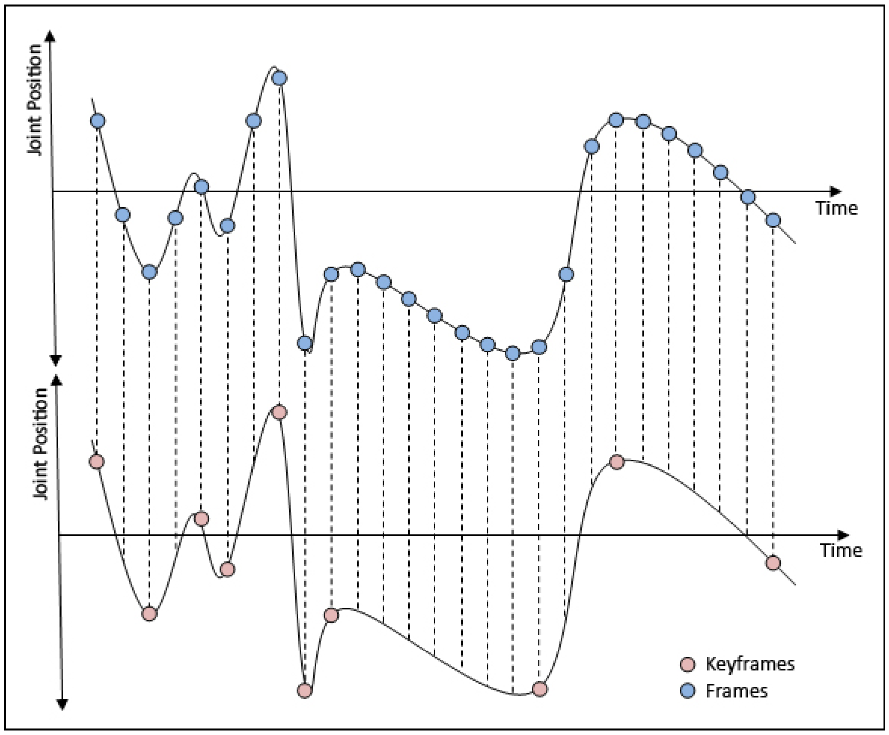


Figure 1.2: The description of a sample joint motion by frames in the upper time axis and keyframes in the lower axis.

At this point the problem can be defined as the search for “good” possible keyframes and their properties such that the overall motion is described as realistic as possible. Finding the keyframes requires to possibly predict any number of keyframes, which can be parameterized since as the number of keyframes gets closer to the number of frames, the motion gets more detailed. As the number of keyframes decreases, the motion would get more and more general. Then the algorithm is required to fit the possible curves between these keyframes such that the motion would follow these curves with as little as possible error on each frame with respect to the original motion. The fitting of the curve requires the curve parameters to be found. When rendering a keyframed animation, the frames between the keyframes are calculated automatically. On the other hand, if the keyframes are identified in the original problem, this would be the reverse problem, if we know the frames between the keyframes, how can the keyframes be found, if they can ever be. By this approach, when keyframes can be generated for various motion capture data, animators can alter the motion easily, undesired noise would be eliminated and the storage space requirements would decrease in a combined way and in one step.

The organization of the rest of the thesis is as follows. Chapter 2 discusses the state of the art on motion capture and keyframe animation, keyframe prediction and extraction from motion capture data, curve fitting and dynamic programming-based keyframe reduction techniques on motion capture data, and curve simplification-based keyframe reduction techniques for motion capture data. Chapter 3 describes the proposed techniques for filtering and keyframe reduction on motion capture data. Chapter 4 gives the visual results and evaluates the performance of the proposed techniques. Chapter 5 concludes the thesis and gives some future research directions. Appendices give the sample motion capture data used in experiments and detailed pseudo-code for the proposed techniques.

Chapter 2

Background

There are some approaches for the process of predicting keyframes: filtering keyframes and curve fitting for the determination of keyframes. This chapter first compares the motion capture animation with the keyframe animation and discusses some approaches for predicting keyframes for a motion data. Then, the approaches for filtering and reducing the number of keyframes is explained. The dynamic programming and curve fitting algorithms are also discussed.

2.1 Motion Capture and Keyframe Animation

Izani et al. compare motion capture animation with keyframe animation [10]. They compared 3 different motions (walking, jumping and running) by using two different methods. Their results include the differences between these two different methods. The motion capture data has more frames than the keyframe method, specifically keyframe method has half the number of frames of the motion capture data. The motion capture method is more complex and it requires more resources to be available. A sample motion capture studio is shown in Figure 2.1. Some principles of traditional animation cannot be achieved using motion capture techniques, such as squash and stretch, overlapping, and exaggeration. On the other hand, keyframing method requires the animator and the tools to be able to clearly create a model animation which also follows the rules of physics. As a result, the method to be used should be chosen according to some criteria such as purpose, cost, practicality, scale and technicality.

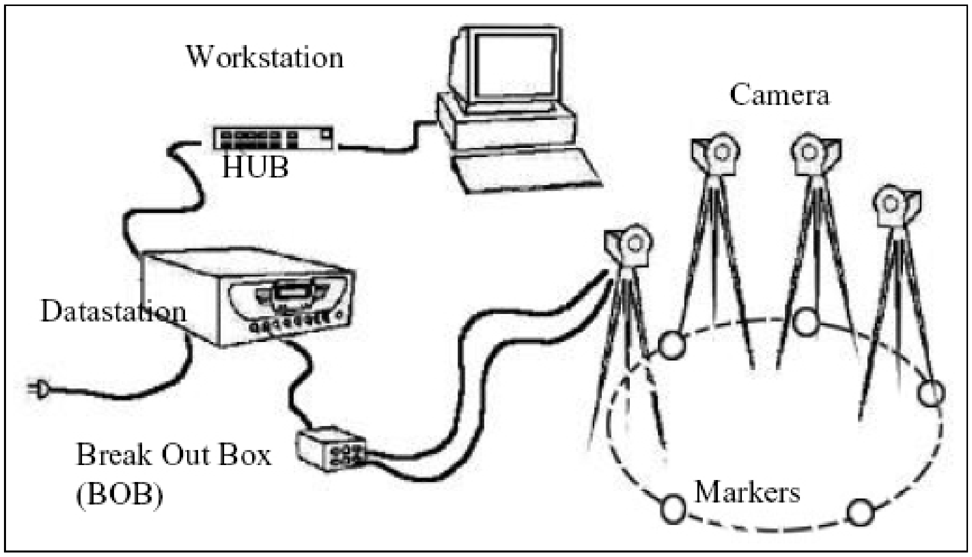
Another work on motion capture and keyframe is done by Geroch et al. [7]. They presented the capabilities that the motion capture systems lack. Usually, motion capture is used in entertainment area and when entertainment is the goal, the budget and the time constraints become critical. The usage of motion capture systems or keyframe animations or even procedural animations where animators have very little power on the animated objects have to be decided on. It is stated that the motion capture is at a very early stage of its development compared to keyframing. Their experience in the field allows them to give some examples of their work and one such outcome is that motion capture is not cheaper or faster then keyframing for the same motion. But they agree that animating realistic human actions using motion capture works very well.

As a result, animators use one of these two approaches according to their needs. Some recent movies are analyzed and compared according to the techniques they use [9]. In this research, the data gathered from the motion capture will be used to create keyframes that can be used by the animators and the entertainment industry, which would enable them to have the benefits of both techniques.

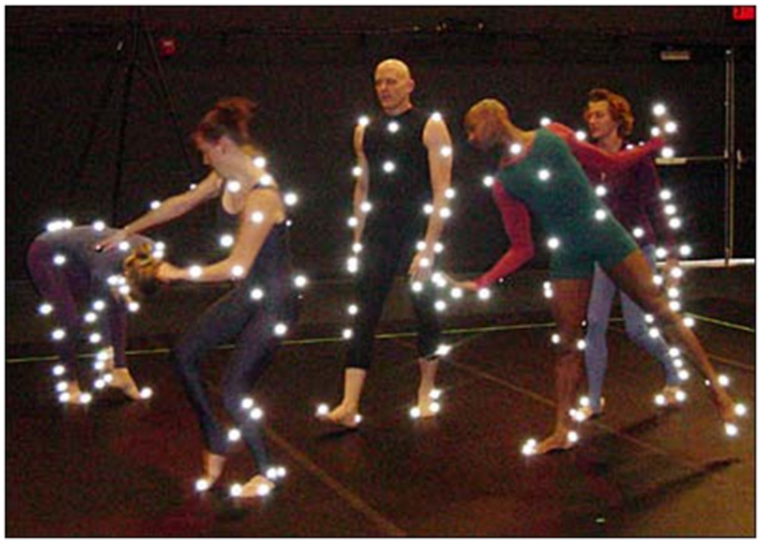
2.2 Keyframe Prediction and Extraction

When the motion captured data is generated, one cannot find information about any possible keyframe without understanding the motion that has taken place in the captured data. There are some research in the field that tries to find how to place keyframes on a motion data. There are also some commercial tools which enable the user/animator to create a motion and decrease the number of keyframes, or create keyframes according to the motion (Figure 2.2).

Terra and Metroyer propose a solution to the problem finding the timings for



(a)



(b)

Figure 2.1: Sample motion capture studio: (a) layout [10]; (b) view [17].

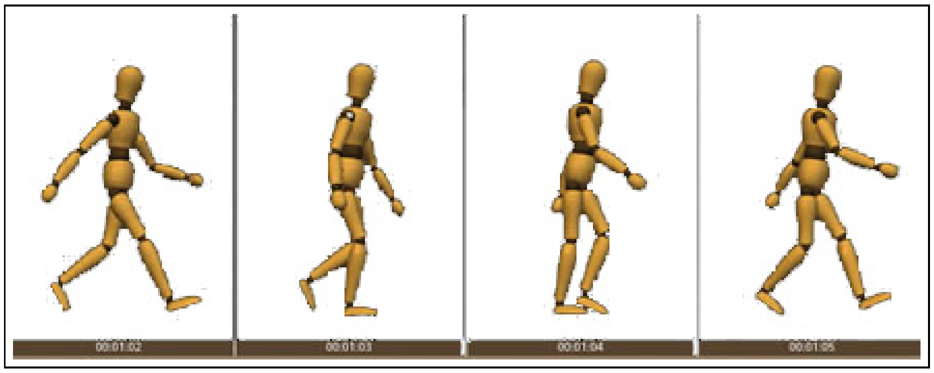


Figure 2.2: Sample keyframed animation [4].

keyframes, which is computationally demanding [25]. Using a performance based approach, they use the “user’s” input for finding timing information of keyframes. Their keyframes are placed on the path by using the user’s input. They re-state that the timing has a huge importance on the understandability of the animation. This approach tries to place the keyframes in their correct positions but the initial keyframes are input from the user. Their approach has an algorithm with 4 steps:

1. Get the keyframe spatial values from the user. At this point, the timing information is ignored and the spatial information is used.

2. Let the user to “act-out” the motion to find out the desired timing. For a selected object on a selected channel, which contains a spatial motion, the user mimics the motion of the object, the generated algorithm captures the timing of the motion.

3. Compute the correspondence between the spatial path and acted-out motion path. At this step, the mapping between the spatial movement and temporal movement is made. The keyframes presented in the first step will be distributed over time. Several approaches can be used for this purpose.

    (a) *Min-max peak features:* the spatial and temporal motion paths are identified and the points where the first derivatives of the curves become zero are located. Then, one-to-one mapping is done between the two motion paths. This would result in keyframes distributed correctly among time. However, this approach is identified as a heuristic one.

   (b) *Curvature analysis:* this finds the curvature parameters and forces them to correspond one-to-one for a scale dependent but rotation independent solution.

   (c) *Contour matching:* this tries to align the two paths by rotating and scaling. Dynamic programming for curve matching is used.

4. Redistribute the keyframe values in time to show the timing of user's performance by using the acted-out motion path. After the correspondence is established between the object motion path and timing path, the keyframes will be adjusted in time while keeping the spatial dimensions constant. During this phase, the curve's control points' tangents at the keyframes must be altered since changing a keyframe in time changes the shape of the curve. The change in the positions are corrected by the changes in the tangents.

Huang et al. propose a method for extracting keyframes, called key probing [8]. They convert the keyframe extraction problem to a constrained matrix factorization problem and they solve it by using least-squares optimization technique. They state that an animation can be represented as $A \approx WH$ where A is the animation sequence, W is the weight matrix and H is the key matrix. The animation is a function of its keyframes, in other words animation sequence is the linear combination of its keyframes. Thus ordinary frames (frames that are not keyframes, or "nonkeyframes") can be found by the linear combination of keyframes. This is different from the general usage of keyframes since in a standard keyframed animation, the inbetweens are usually the interpolation of the keyframes. Their technique consists of an iterative approach with the following operation in each iteration:

$$u \leftarrow min_{v \in A_{nkey}} \left\| A_{nkey} - A_{nkey} H^{\phi} H \right\|^2,$$

$$H^{i+1} \leftarrow H^i \oplus A^u_{nkey},$$

$$A_{nkey} \leftarrow A_{nkey} \ominus A^u_{nkey},$$

$$W^{i+1} \leftarrow A_{nkey} H^{\phi i+1},$$

where i is the number of iterations and the number of keyframes selected, W is the weight matrix, H is the key matrix, A_{nkey} is the animation sequence matrix that contains nonkeyframes, H^{ϕ} is the pseudoinverse of H such that $A_{nkey}H^{\phi}$ denotes weight matrix W, u is the index number of the current keyframe, $\oplus$ is the set union operator, $\ominus$ is the set difference operator. If after this iterative keyframe extraction process the results are not satisfying, the same process can be applied to the newly found keyframes such that the same algorithm is used on the result.

Park proposed a method to clone a motion data from one model into another [19]. While doing this operation, a small number of representative postures called key-postures were extracted. It is required that the key-postures define nearly all the postures of the animation and are as small as possible in terms of the number of key-postures. Similarly, as in [8], their definition of key-postures is that their linear combination enables one to get all the postures of the animation. Their algorithm for key-posture extraction can be summarized as follows:

1. Find an initial collection of clusters C from given sample postures.
2. For each cluster, the key-posture is selected as the one that is near to the center of the cluster, for finding initial key-postures.
3. In an iterative manner, augment every cluster by adding a new cluster at each loop, until every sample posture can be expressed as a linear combination of the chosen key-postures with a given threshold ϵ.
4. While doing the loop, find the maximum error value between the postures and the reconstructed postures. If maximum error is smaller then some threshold, this set of chosen key-postures can be used to generate all sample postures within some small error range. Otherwise, the cluster collection is augmented with the posture that gives the highest error value and the loop starts again. According to their experimental results, a threshold of 2-5 % is the optimum value.

2.3 Curve Fitting and Dynamic Programming Approaches

There are many dynamic programming cases in the area such as [2, 11, 16]. Dynamic programming is used when searching for a solution with minimal error according to certain criteria. In [1], dynamic programming is used to solve some variation problems in computer vision. They state that the dynamic programming ensures global optimality of the solution. Dynamic programming allows some hard constraints to be held while maintaining the natural and straightforward structure of the motion. Dynamic programming approach makes sure that the results stay in stabilized form. It helps to solve optimization problems with minimum energy or error according to the problem definition.

There is no generic research up to date that shows how dynamic programming can help to find keyframe positions that minimizes the error according to the action motion capture data. In the following chapter, the dynamic programming approach will be used for fitting curves on the motion capture data. With the help of keyframe prediction techniques, this method will try to find and fit the keyframes that has minimum error that is compared to the original data, and while viewing there would be minimal difference between the motions.

Curve fitting algorithms are available in various areas of computer science. They are usually used to fit curves over fonts and over image contours for computer vision. One of the many such approaches can be found in Baudel's research [3]. He proposes to create splines with more friendly gestures such as strokes and shapes, and provides an interaction technique. Other researches include [22, 23] whom try to cover some different approaches for animation control and using fitted curves representing the animation.

There are also more general methods that can be used for fitting curves. This process may require some constraints such as in [6] or curve fitting might be done using piecewise parametric cubics [21] or conic splines [20].

2.4 Curve Simplification Approaches

Curve Simplification algorithms are mainly used to simplify or summarize motion information. For instance, Kraus describes how to generate new motions by using existing motion data [12], and while generating new motions, he uses Lowe's algorithm [14] for the reduction of motion capture data.

Lim and Thalmann propose that human motion data can be analyzed for key-postures by using curve simplification methods [13]. Specifically, they again use Lowe's algorithm [14] for this process. Their curve simplification method is based on the fact that points on the motion curve graph can be grouped together and simplified. They suggested that with 0.5 % error tolerance, the data size has been dropped to 22.4 % with this method.

2.5 Summary

Table 2.1 shows the comparisons of different approaches according to certain criteria.

Table 2.1: Summary of different algorithms and approaches.

Approach	Extracts Keyframes	Keyframe Time Adjustment	Absolute Results	Repeatable
Terra [25]	No	Yes	Yes	No
Huang [8]	Yes	Yes	No	Yes
Park [19]	Yes	Yes	No	Yes
Lim [13]	Yes	Yes	No	Yes
Dynamic Prog.	No	Yes	Yes	No

Chapter 3

Keyframe Reduction for Motion Capture Data

We propose two different approaches to the keyframe reduction problem for motion capture data: curve fitting with dynamic programming and curve simplification.

3.1 Curve Fitting with Dynamic Programming

In this approach, dynamic programming will be used to fit a curve on the motion capture data. Motion capture data contains the positions or rotations of each and every joint that is present on the stage over time (which is also represented by the frames). So a motion of a joint, its position or rotation can be defined on a graph (position or rotation versus time or frame) such as in Figure 3.1. The algorithm will try to fit a curve on each such graphs for every joint. Before going on the details of the algorithm, the curve definition will be presented.

Hermite curves will be used on this algorithm because of their properties on the control points. The control point of a Hermite curve contains the positions and the tangents at the end points. Similarly, animators who work on keyframed

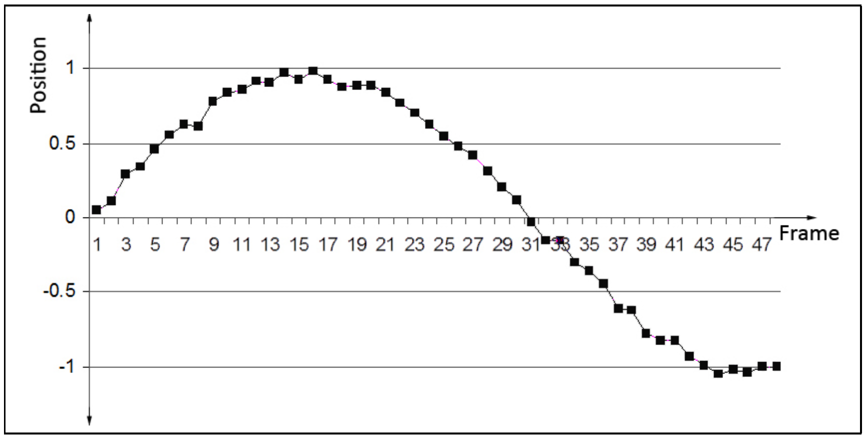


Figure 3.1: A Motion Curve for a Joint.

animations use keyframe points as the position and continuity indicators, which means they use the tangent of the motion curve while starting the motion (starting keyframe) and ending the motion (ending keyframe). Thus, the control of the continuity is important and the best curve that fits this definition is the Hermite curve. A Hermite curve $v(s)$ between the control points i and $i+1$ can be described as;

$$v(s) = \begin{bmatrix} (s-i)^3 & (s-i)^2 & (s-i) & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_i \\ p_{i+1} \\ r_i \\ r_{i+1} \end{bmatrix}$$

where p_k represent the positions and r_k represent the tangents at keyframe k, which will be called the control vectors of a Hermite curve. Such an Hermite curve can be visualized as in Figure 3.2.

At any point, this Hermite curve can be changed into a Bézier curve as described in [5]. This operation is needed whenever a point on the curve is requested. Since the control points of a Hermite curve define the curve's shape, a specific point on a curve is needed at certain point in time (or in a certain frame number). Also since a Bézier curve can be recursively subdivided into two Bézier curves, the value of a certain point in time can be achieved using this approach [5]. This operation is illustrated in Figure 3.3. The subdivision algorithm enables one to

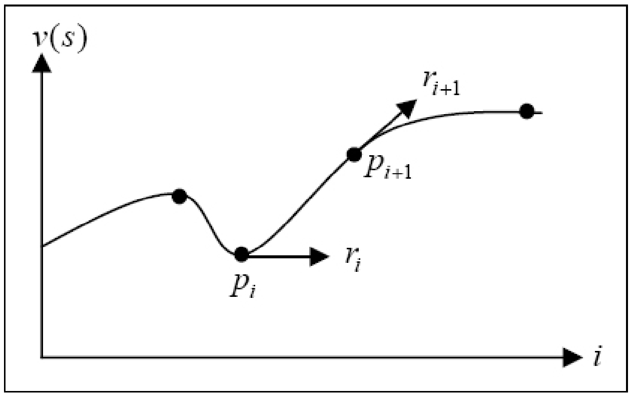


Figure 3.2: Hermite curve is defined by its control points and tangent vectors.

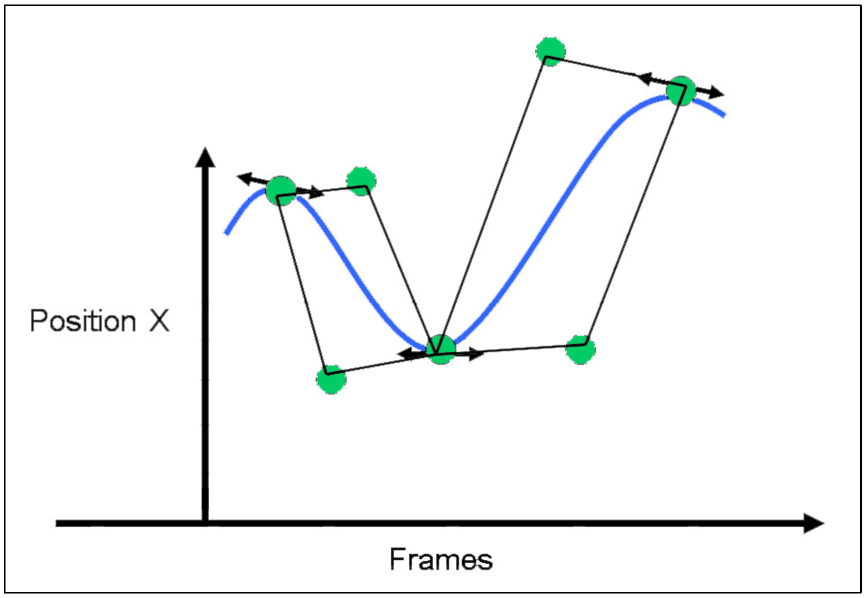


Figure 3.3: A subdivision for a Bézier curve.

subdivide the curve as much as he/she wants; it will be subdivided until the requested accuracy is achieved. In our case, the algorithm can be summarized as follows (detailed pseudo-code is given in Appendix B.1):

1. Find initial keyframe estimates, by predicting p_i, p_{i+1}, r_i and r_{i+1} for every i. This step involves predicting possible keyframes, which was discussed in the previous chapter. It can be a complex operation such as trying to understand the motion and subdividing it into its components or it can be as simple as placing the keyframes on the points that the first and second derivatives of motion graph are both zero. This step involves the

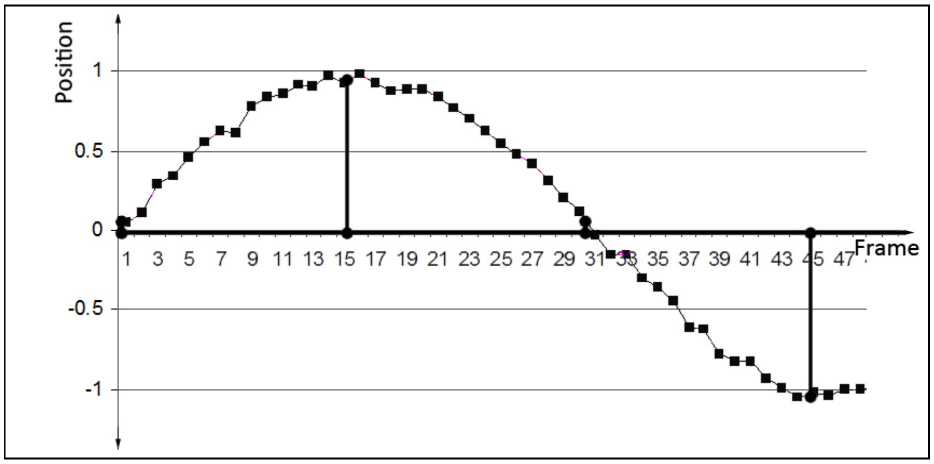


Figure 3.4: Finding/predicting initial keyframe positions.

decision of the number of keyframes and their initial positions, hence for the smoothness or the amount of detailed motions of the animation, is done at this step. With the example Figure 3.1, assume that the keyframes are predicted at the positions that are shown in Figure 3.4. For every joint, its position or rotation data is gathered as a motion capture data. Whether its position or rotation, it will have three dimensions (x, y and z axes). For every axis, this keyframing operation is done and for every axis a graph with two dimensions can be constructed such as time versus the joints position on that axis.

2. Determine search spaces around the predicted keyframes for all vectors p_i, p_{i+1}, r_i and r_{i+1} for every i. Since each vector has x and y components (y-value can be described as the joint's position in p_i while x value represents the time, and tangent value can be represented by the y value in r_i and x value represents the time), the search spaces are formed around these points. In Figure 3.5, these search spaces are visualized. In the first step, the control points are predicted roughly, in the later steps they will be searched for their best positions in the search spaces that are found in this step. These search spaces reside around the original points, but since each curve segment (in the overall curve, between two control points) has infinitely many number of positions, dynamic programming will be used to find the best possible position. The search spaces will be assumed to contain

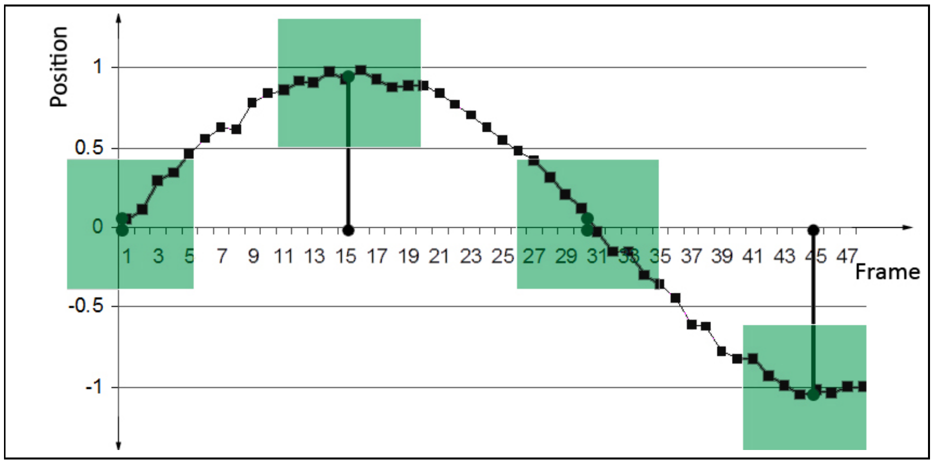


Figure 3.5: Search spaces around control points.

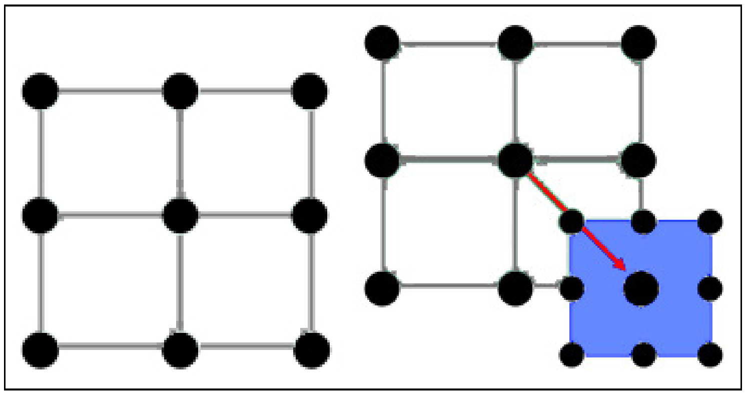


Figure 3.6: Search space states for each keyframe point. When a state is selected as the best, a sub-search space will be constructed around that point.

only 9 possible states as shown in Figure 3.6. The best of these 9 states will be found and recursively the size of the search space will be decreased and another 9 set of states will be generated. Each time, the state that gives the best result will be selected and the algorithm will continue recursively until certain threshold is reached. At a certain point in time, two control points will have 2 dimensions for position and rotation, and also they will have 3 different possible states (3-x states, 3-y states). Thus, it will create $3 \times 3 \times 3 \times 3 = 81$ different states for one curve segment. For one joint and for one axis on that joint, a curve segment will contain 2 dimensions (position and rotation), so $81 \times 81 = 6561$ different possible combinations are present that can affect the shape of the curve.

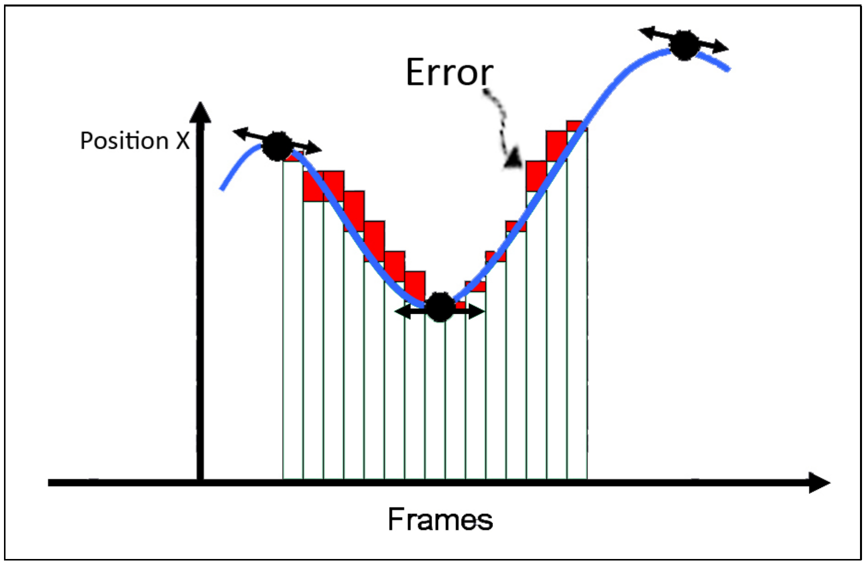


Figure 3.7: Error between the original data and current Hermite curve.

3. Compute the mean squared error for each combination of p_i, p_{i+1}, r_i and r_{i+1} in the search space. The mean squared error (MSE) can be found as the mean of the squares of the difference between the value of the Hermite curve at a frame and the value of the motion capture data at that frame (Figure 3.7). The value of the Hermite curve at some point in time can be found with Bézier curve conversion and Bézier curve subdivision algorithm as described recently. For each combination of p_{i+1} and r_{i+1}, the best values of p_i and r_i (that gives the minimum error) are saved.

4. When the last segment of the curve is reached, find the total mean squared error for each combination of p_{i+1} and r_{i+1}. To calculate total mean squared error, traverse the curve backwards and use the best (minimum) previous values obtained in step 3.

5. Pick the overall path of control vectors that results in the minimum mean square error in step 4.

6. Assume that the values p_i, p_{i+1}, r_i and r_{i+1}, for all i, obtained in step 5 are the updated initial values and repeat steps 2-6 until a predetermined accuracy in x and y components is reached.

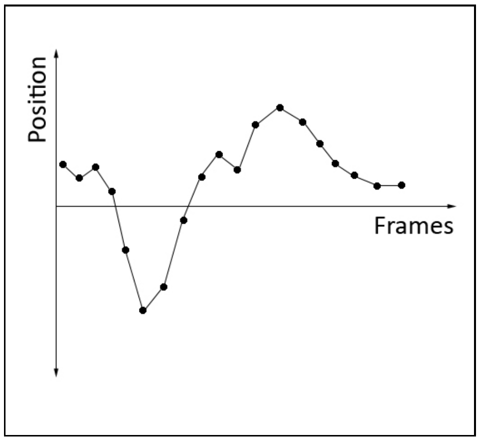
This algorithm employs dynamic programming to fit a Hermite curve to motion capture data in mean squared error sense.

3.2 Curve Simplification

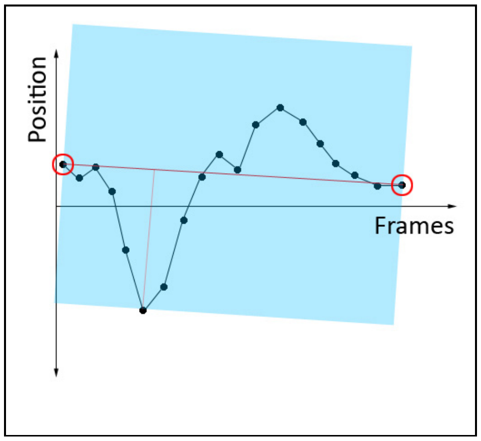
The algorithm on curve simplification approach uses a step by step approach. Initially the motion graph will be assumed to have only one curve segment (two keyframes, at the beginning and at the end). The highest error points in time between these two keyframes will be used to subdivide the curve into two curves. The algorithm can be summarized as follows (detailed pseudo-code is given in Appendix B.2):

1. Create 2 initial keyframes, one at the beginning and at the end of the animation.
2. Assuming that there is a virtual line that connects these two keyframes for every other frame that stays between these two frames, calculate a minimum distance between the position of the joint at that frame and the virtual line. The maximum value of this distance is the maximum error value of the curve segment if there were no other keyframes present. In Figure 3.8(a), a sample motion graph is given and in 3.8(b) the initial step is visualized. So, a keyframe at the maximum error value would decrease the total error value which is the goal.
3. If the ratio of the error distance to the distance between the beginning and ending keyframes is higher then a specified threshold, stop further subdividing this interval. Otherwise, create a new keyframe at the point where the highest error occurs.
4. Subdivide the current state into two smaller problems, a segment between the beginning keyframe and newly created keyframe, and another segment between the newly created keyframe and the ending keyframe. Assume that the new segments have only 2 keyframes (beginning and ending) and restart the algorithm from step 2 for both segments.

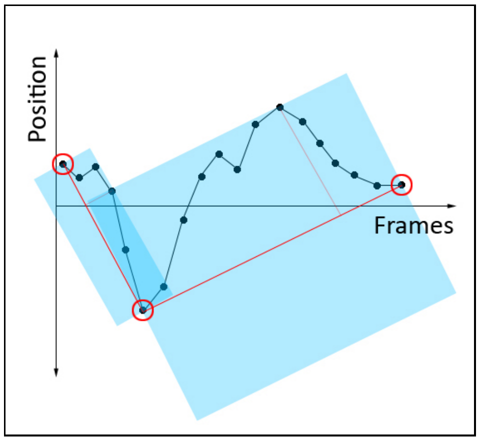
For the sample data, this algorithm will continue as shown in Figures 3.8(a-d). In 3.8(d), the values that will be used on the ratio check is also shown as w for width and h for height. The recursive approach by subdividing the curve segments will stop at some point where all the ratios satisfy certain conditions. This threshold can be modified or parameterized according to the time such that at certain parts of the animation, the detail is conserved where as in the other places, the motion is smoothed. When all the instances of the algorithm stops, keyframes will already be placed on the motion path. At this point, another pass over the keyframes will find the optimum values for the tangent values at the keyframe points which minimizes the total error and preserves continuity. After this step, the result is illustrated in Figure 3.8(e).



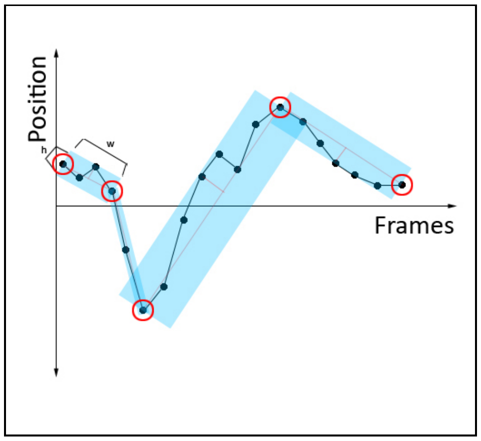
(a)



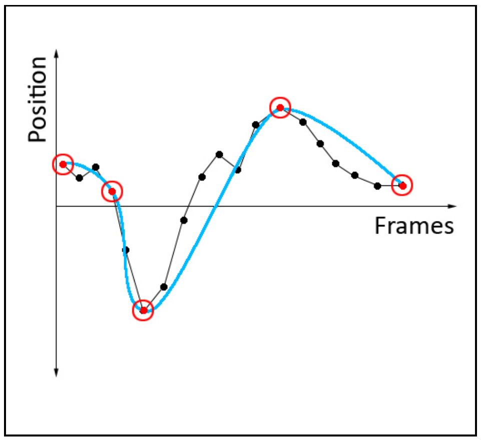
(b)



(c)



(d)



(e)

Figure 3.8: The application of the curve simplification algorithm: (a) the initial curve; (b) the first step; (c) the second step; (d) the third step; (e) the final result.

Chapter 4

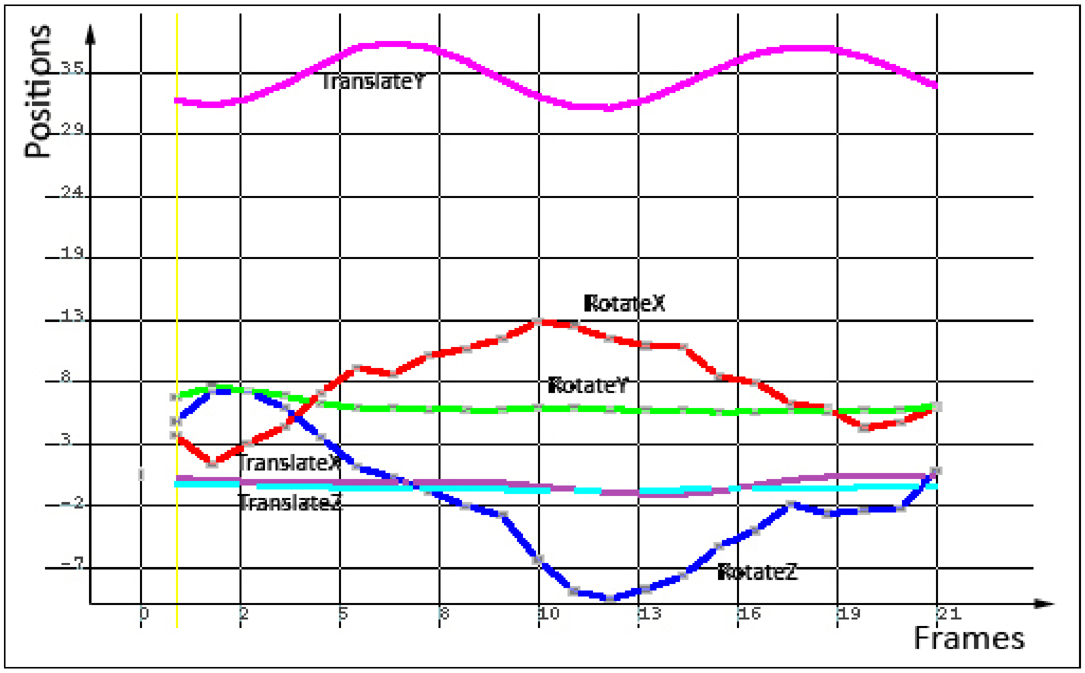
Experimental Results

4.1 Results

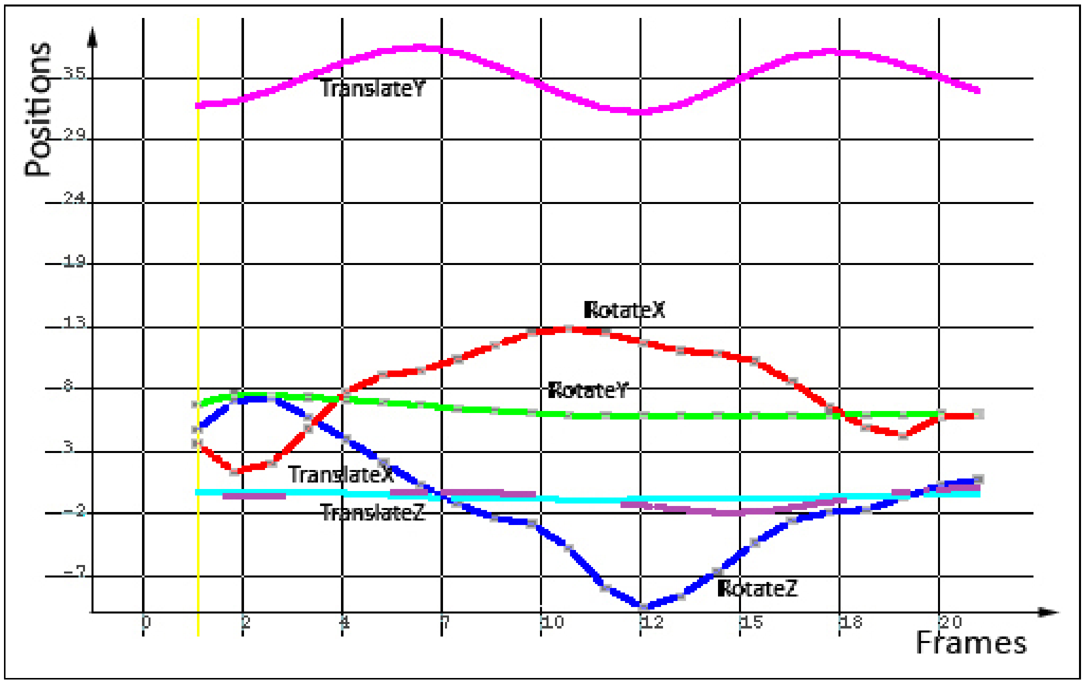
4.1.1 Curve Fitting with Dynamic Programming

The results presented here are the results of the algorithm that is run on two different motion capture data. The first one has small number of frames with relatively simple motion, running. There are 22 frames present on the motion capture data. Figure 4.1 shows the motion graph of the chest joint for this animation. In Figure 4.2, some sample frames are shown from the animation. For this joint there are 6 channels (position x-y-z and rotation x-y-z), in the overall model there are 84 channels. The number of frames per channel in the original motion capture data is 22. The average number of keyframes per channel in the resulting animation data is 8.

The second one has 1409 frames which is the motion for a human that walks, sits, then takes a cup of tea and drinks it. Figure 4.3 shows the motion graph of the head joint for this animation. In Figure 4.4, some sample frames are shown from the animation. For this joint there are 6 channels (position x-y-z and rotation x-y-z), in the overall model there are 84 channels. The number of frames per channel in the original motion capture data is 1409. The average number of keyframes per channel in the resulting animation data is 459.



(a)



(b)

Figure 4.1: Chest position/rotation graph for running motion: (a) original; (b) simplified.

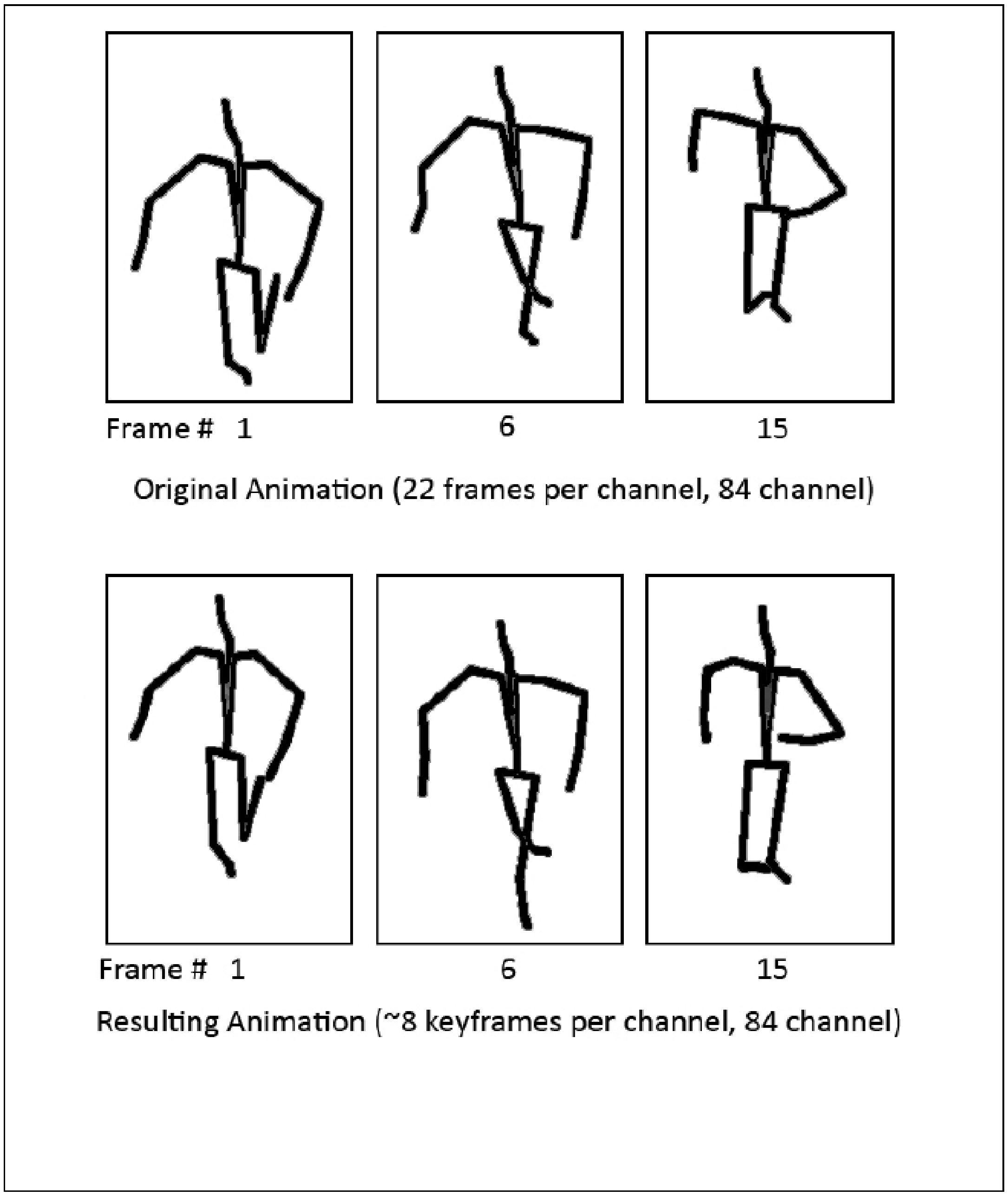
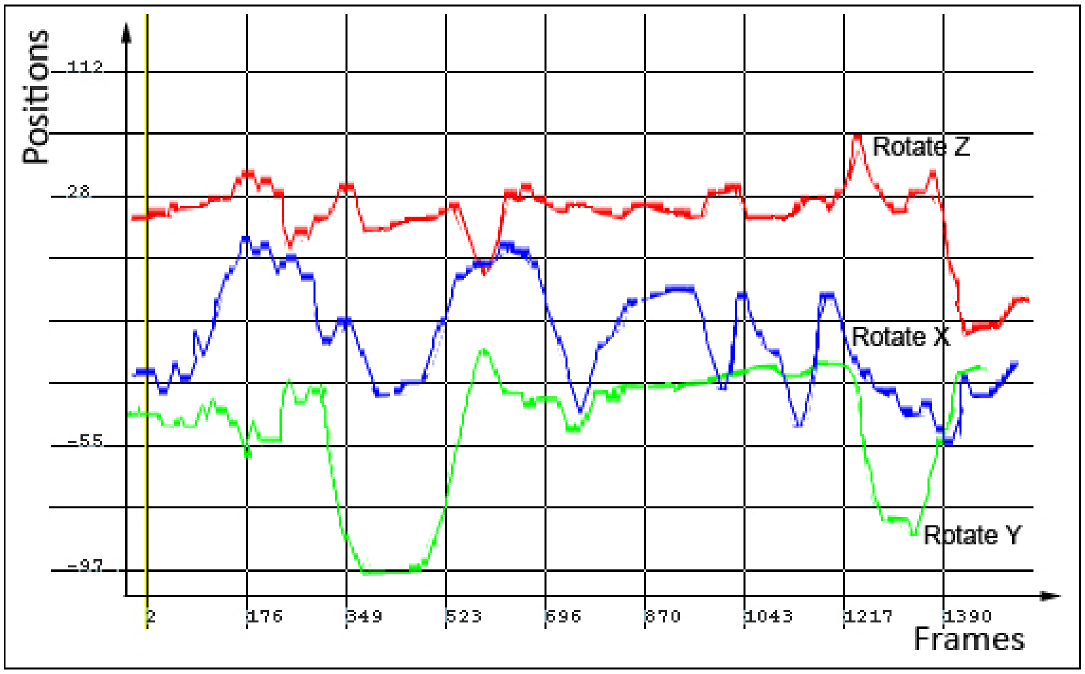
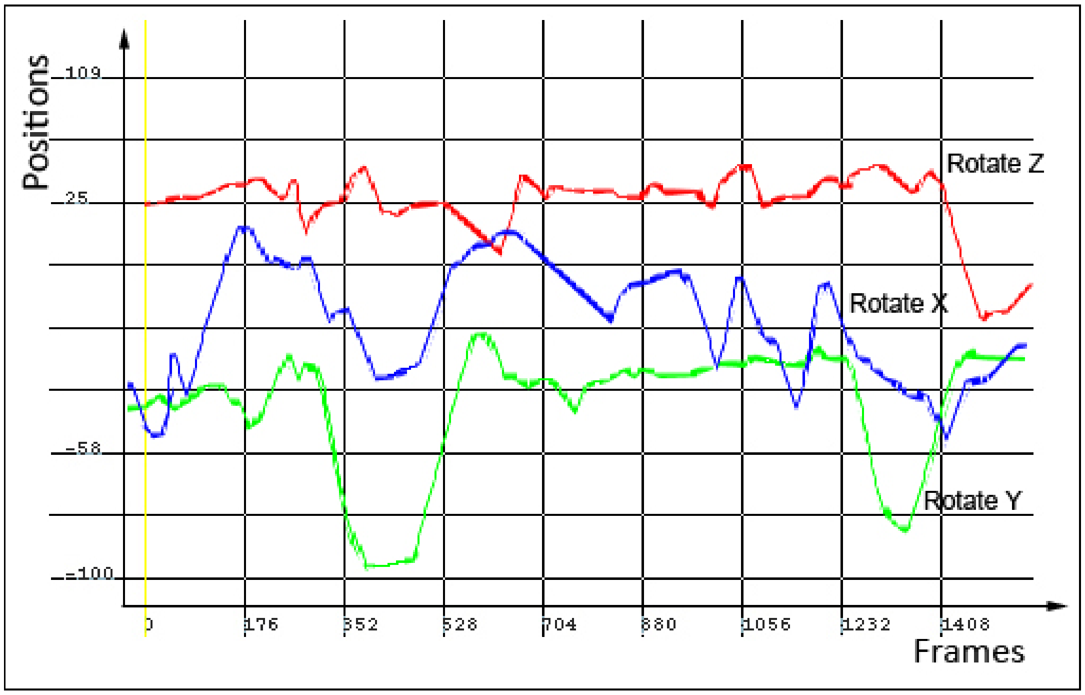

Frame # 1
6
15
Original Animation (22 frames per channel, 84 channel)
Frame # 1
6
15
Resulting Animation (~8 keyframes per channel, 84 channel)

Figure 4.2: Sample frames from original running motion and the resulting running motion.



(a)



(b)

Figure 4.3: Head rotation graph for drinking tea motion: (a) original; (b) simplified.

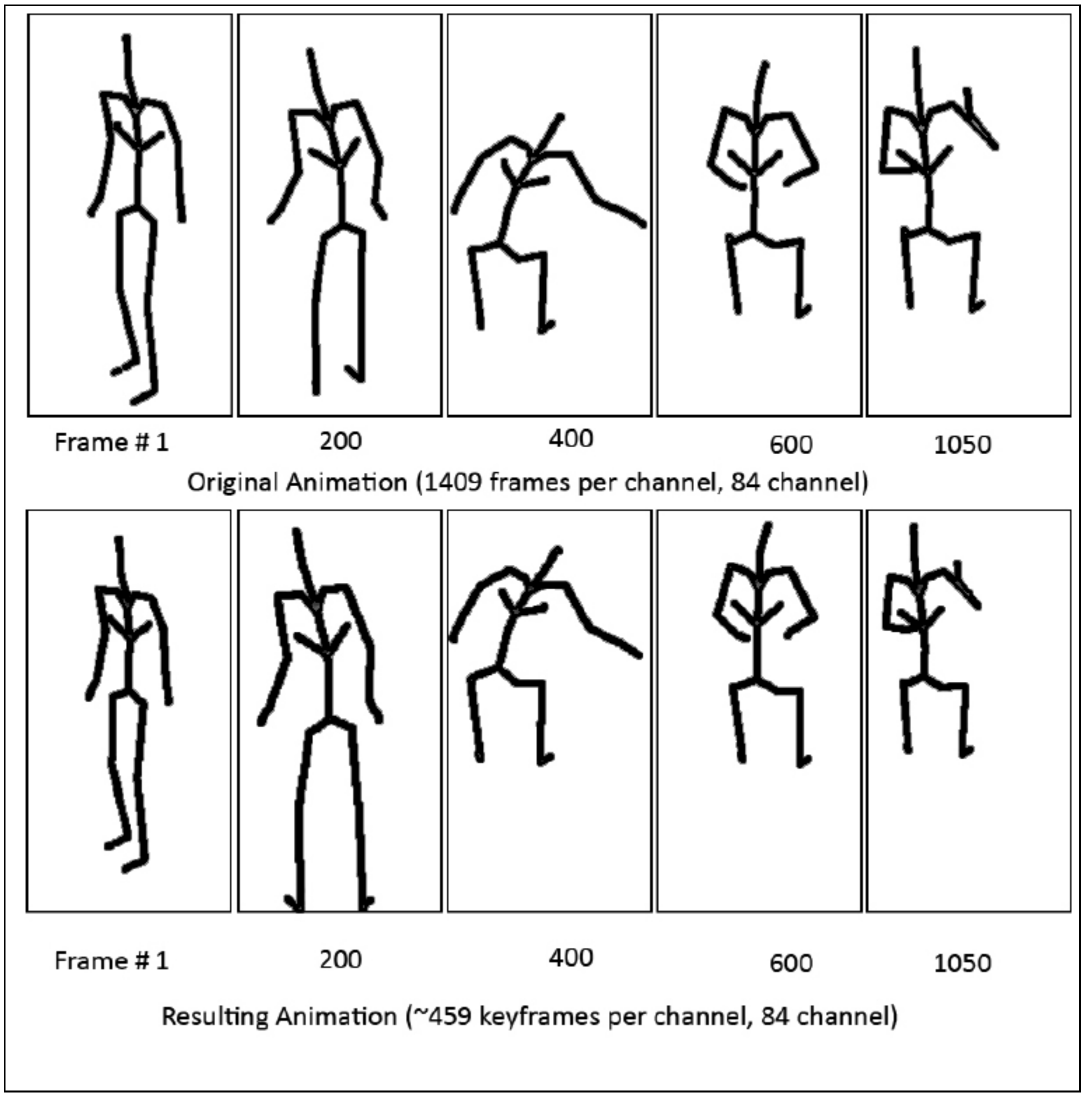

Frame # 1
200
400
600
1050
Original Animation (1409 frames per channel, 84 channel)
Frame # 1
200
400
600
1050
Resulting Animation (~459 keyframes per channel, 84 channel)

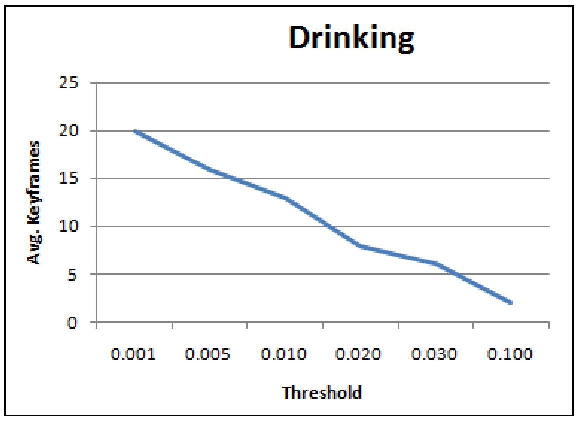
Figure 4.4: Sample frames from original drinking tea motion and the resulting drinking tea motion.

4.1.2 Curve Simplification

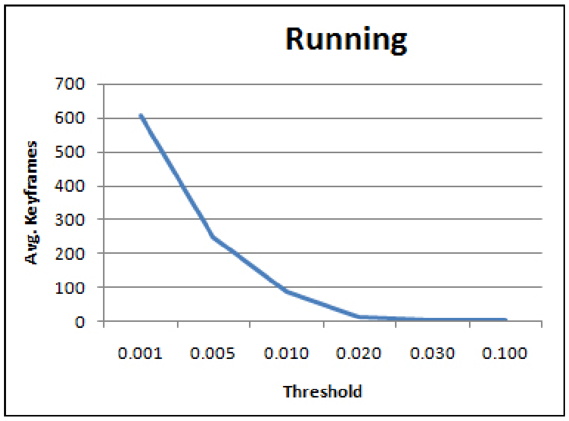
Table 4.1 shows the results of applying the algorithm on sample data. These results can also be visualized in Figures 4.5 (a) and b).

Table 4.1: Results of number of average keyframes per channel on several runs.

	Curve Simplification	
Threshold	Running	Drinking
0.001	608	20
0.005	246	16
0.010	85	13
0.020	11	8
0.030	3	6
0.100	2	2



(a)



(b)

Figure 4.5: Threshold value vs. number of keyframes for (a) drinking animation and (b) running animation.

4.2 Evaluation

4.2.1 Objective Evaluations

Table 4.2 shows the comparisons of the two different approaches according to the empirical results.

Table 4.2: Summary of the two approaches based on empirical results.

	Curve Fitting with Dynamic Programming		Curve Simplification	
Properties	Running	Drinking	Running	Drinking
# of frames	22	1409	22	1409
Channels	84	84	84	84
Avg. # of Keyframes	8	459	6	246
Frame Reduction Ratio	36 %	32 %	27 %	17 %

Average number of keyframes are found by dividing the total number of keyframes to the number of channels. The algorithms are executed for every joint and end-effector separately. For different channels the keyframes are not aligned on time axis, thus one model might have different number of keyframes for every channel. If such requirements are present, a post processing algorithm might be needed for aligning the keyframes on different channels and having a keyframe for every channel. The execution times are tabulated in Table 4.3. The execution times do not include the loading of the data. A graph showing the relation between execution time and movie length is given in Figure 4.6. The error threshold in curve simplification algorithm is adjusted to have the resulting animation of similar quality with the dynamic programming algorithm.

Table 4.3: Summary of the execution times of two approaches.

Animation	Curve Fitting with Dynamic Programming	Curve Simplification
Running	1,959 ms	822 ms
Drinking Tea	6,923 ms	5,445 ms

*Execution times are calculated on a Pentium Centrino 1.7 GHz processor.
**Loading times are excluded.

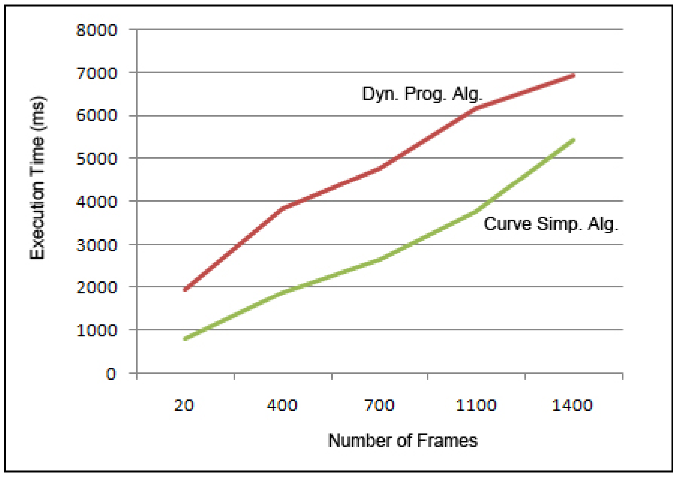


Figure 4.6: Movie length vs. execution times for each algorithm.

Some results are discussed and compared below:

- While the previous approaches achieve up to 80 % decrease in the size of the animations and number of frames (or keyframes), these techniques present new algorithms which brings similar results that have frame reduction percent again around 75-80 %. The curve simplification method provides better results both in terms of the error ratio and the number of keyframes.
- With the dynamic programming approach, the keyframes cannot be reduced in number after the initial estimates are predicted. Whereas in curve simplification approach, the number of keyframes is directly proportional with the number of times the algorithm iterates, thus it can be constrained or limited.
- Error threshold cannot be easily controlled on dynamic programming approach as the curve simplification method. Dynamic programming will find the absolute best positions/tangents for the keyframes but if the initial keyframe estimator brings up keyframes that contain high number of error ratio, the dynamic programming would not be able to correct them (because of the limited search spaces). On the other hand, curve simplification method finds the keyframes on the fly and the total error ratio can be set while continuing the algorithm, thus it will allow to change the desired error ratio to get more or less keyframes.

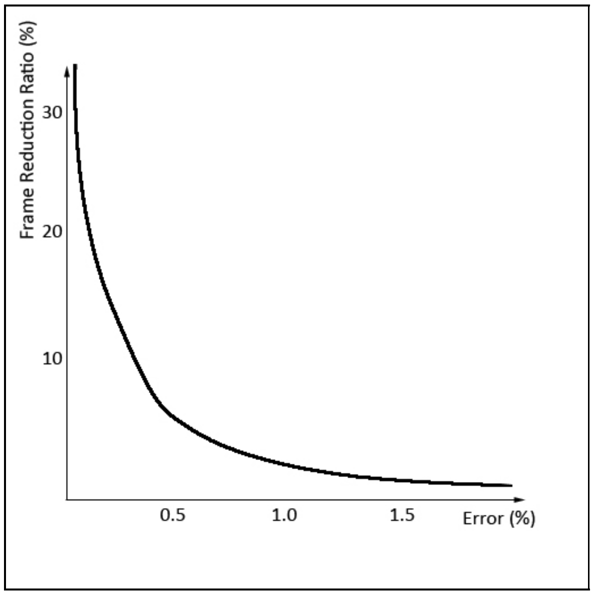


Figure 4.7: Error percent vs. frame reduction ratio.

- Error percentage is inversely proportional to the Frame reduction ratio (Figure 4.7). As the number of keyframes decreases, error ratio increases; and vice versa.
- Both methods successfully reduces the number of keyframes thus eliminating the noise factor.
- Curve simplification method can be used on already "keyframe-reduced" animations to further decrease the number of keyframes. On the other hand dynamic programming method can be used to clearly improve the positions and tangents of the keyframes to better map the given original motion data.
- Curve simplification method requires special attention on the rotation channels since they contain a limited interval of values, as the animation size increases the possibility of having a good keyframe distribution decreases.

4.2.2 Subjective Evaluations

The visual quality of the results produced by both approaches are quite satisfying. Although some differences between the original animation can be noticed, it can be said that most of the animation is as realistic as possible. Some comparisons between the methods can be found below.

- The visual results obtained using the curve simplification method is slightly better then its counterpart. Dynamic programming's inability to be applied multiple times and thus inability to decrease the error value after some point limits its approach to the goal (reality).
- Curve simplification method can be used to separate the motions into categories and distinguish between the motions that require more smooth animations (for eliminating the noise) or that require more detailed animations to preserve the complexity of the animation. Curve simplification method allows the user to select different error thresholds for different parts of the animation. Thus it allows diversity among the motion according to the needs of the user.
- Both methods successfully eliminate any possible noise that may be resulting from the motion capture system.

Chapter 5

Conclusion

This research aimed to cover a solution to merge the advantages of two different animation techniques while decreasing the space requirements and providing noise elimination. In the previous chapters, the state of the art is analyzed, the up to date research of keyframing and motion capture animation are presented. The algorithms to offer reliable solutions are proposed and implemented. The results are also given with different sample inputs. Since the outcome of such solutions would generally be used for computer aided entertainment, the evaluation of the results are given both objectively and subjectively.

In the field of animation in computer graphics, keyframing techniques have been developed to its current mature state. However, the motion capture technology is a newly growing and rapidly developing area. Both techniques have their own advantages and disadvantages. A keyframed animation can use all the animation techniques as the animator desires, even if they are impossible in a real world. However, a keyframed animation becomes a hard and tiring technique when trying to simulate a real world situation. Motion capture animation technique has the ability to greatly mimic the real world in a virtual and confined environment, but it lacks some principals of the animations as well as the easiness of the modification of animations with respect to a keyframed animation. Keyframed animation requires less storage space but it will always lack some motion data as compared to the motion capture data.

Recent researches presented some approaches to solve these problems. Some considered which technique to use on which situations, some others considered how to use both techniques on many situations. Many approaches focused on keyframe prediction or key posture extraction techniques to successfully understand a summary of the motion, thus use the advantages of motion capture data with keyframe animation. Techniques included diverse algorithms including curve fitting, curve simplification, dynamic programming.

Two of the algorithms, which are mentioned beforehand, are used to obtain a solution for the problem. Although dynamic programming with curve fitting implementation was never used before, it presented some challenging results with respect to other algorithms. Curve simplification results were slightly better than the dynamic programming algorithm. Both methods tried to create keyframes out of motion capture data, and they tried to obtain the most accurate curves that decrease differences from the original motion. Again both methods were successful on decreasing the number of keyframes for a motion capture animation and filtering any jitter that may be produced by the motion capture system. Curve simplification method had some advantages over the dynamic programming approach such as the ability to control the number of keyframes and the value of the error percentage with respect to the original motion.

If a motion captured data is available, and the keyframes are desired on the situation, filtering and keyframe reduction of motion capture data using curve simplification algorithm is the best alternative. It provides decreasing of the number of keyframes on a motion capture data to 20% by only having a 0.5% error ratio. It enables to have a good compression or keyframe reduction, noise-free and easily modifiable motion while preserving the realistic components of the animation.

Bibliography

[1] Amini, A. A., Weymouth, T. E., Jain, R. C., "Using Dynamic Programming for Solving Variational Problems in Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. 12, No. 9, pp. 855-867, Sept 1990.

[2] Baird, L.C., "Function Minimization for Dynamic Programming Using Connectionist Networks," *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics,* Vol. 1, pp. 19-24, Chicago, IL, U.S.A., 1992.

[3] Baudel, T., "A Mark-based Interaction Paradigm for Free-hand Drawing ," *Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology,* California, U.S.A., pp. 185-192, 1994.

[4] Carrara Studio 2 Sofware, *http://www.completeanimation.com/techstuff/software/carrara2.html.*

[5] Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F., *Computer Graphics: Principles and Practice, 2nd Ed. in C,* Addison-Wesley, 1995.

[6] Fowler, B., Barrels, R., "Constraint-Based Curve Manipulation," *IEEE Computer Graphics and Applications,* Vol. 12, No. 5, pp. 43-49, September 1993.

[7] Geroch, M. S., Hirsch, E., Staveley, J., Tolles, T., Helfer, B., Varadarajan, S., "How Does Motion Capture Affect Animation?," *ACM SIGGRAPH Conference Abstracts and Applications,* Texas, U.S.A, pp. 103-104, July 2002.

[8] Huang, K., Chang, C., Hsu, Y., Yang, S., "Key Probe: A Technique for Animation Keyframe Extraction," *The Visual Computer,* Vol. 21, No. 8-10, pp. 532-541, September 2005.

[9] Izani, M., Aishah, Eshaq, A. R., Norzaiha, "Analysis of the Keyframe Animation and Motion Capture Case Studies," *Proceedings of the Student Conference on Research and Development (SCOReD),* Putrajaya, Malaysia, pp. 177-182, 2003.

[10] Izani, M., Eshaq, A. R., Zainuddin, N., Razak, A., "A Study on Practical Approach of Using Motion Capture and Keyframe Animation Techniques," *Proceedings of the Eight International Conference on Information Visualisation (IV'04)* pp. 849-852, 2004.

[11] Jacobson, D. H., Mayne, D. Q., "Differential Dynamic Programming," *Elsevier,* New York, NY, 1970.

[12] Kraus, M., "Human Motion and Emotion Parameterization," *Proceedings of the Conference of the Central European Seminar on Computer Graphics* 2004.

[13] Lim, I. S., Thalmann, D., "Key-posture Extraction out of Human Motion Data by Curve Simplification," *Proceedings of the 23rd Annual EMBS International Conference,* Istanbul, Turkey, 2001.

[14] Lowe, D. G., "Three-Dimensional Object Recognition from Single Two-Dimensional Images," *Artificial Intelligence,* Vol. 31, No. 3, pp. 355-395, March 1987.

[15] Mealing, S., "The Art and Science of Computer Animation," *Cromland,* 1995.

[16] Morimoto, J., Christopher, G. A., "Minimax Differential Dynamic Programming: An Application to Robust Biped Walking," *Advances in Neural Information Processing Systems,* **15**, Cambridge, MA, 2002.

[17] Motek - Motion Technology, *http://www.e-motek.com.*

[18] Mukai, T., Kuriyama, S., Kaneko, T., "Keyframe Animation of Virtual Humans via Motion Data Learning," *Systems and Computers in Japan,* Vol. 37, No. 5, 2006.

[19] Park, M. J., Shin, S. Y., "Example-based Motion Cloning," *Computer Animation and Virtual Worlds,* **15**, pp. 245-257, 2004.

[20] Pavlidis, T., "Curve Fitting with Conic Splines," *ACM Transactions on Graphics,* Vol. 2, No. 1, pp. 1-31, January 1983.

[21] Plass, M., “Curve Fitting with Piecewise Parametric Cubics,” *Proceedings of the 10th ACM Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH’83)* pp. 229-239, 1983.

[22] Schneider, P. J., “An Algorithm for Automatically Fitting Digitized Curves,” *Graphics Gems*, Academic Press, San Diego, CA, pp. 612-626, 1990.

[23] Snibbe, S. S., “A Direct Manipulation Interface for 3D Computer Animation,” *Computer Graphics Forum,* Vol. 14, No. 3, pp. 271-283, 1995.

[24] Terra, S. C. L., Metoyer, R. A. “A Performance Based Technique for Timing Keyframe Animations,” *Graphical Models,* Vol. 69, pp. 89-105, 2007.

[25] Terra, S. C. L., Metoyer, R. A., “Performance Timing for Keyframe Animation,” *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation,* pp. 253-258, 2004.

Appendix A

Motion Capture Data

The motion capture data for the running and drinking animations are given in Biovision Hierarchical (BVH) data file format. Because of space sonstraints, the data for the two initial frames are given for each animation.

A.1 Running Animation

```
HIERARCHY
ROOT Hips
{
  OFFSET  0.00  0.00  0.00
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation
   Yrotation
  JOINT Chest
  {
    OFFSET   0.000000   4.570000   0.000000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Neck
    {
      OFFSET   0.000000   17.620001   0.000000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT Head
      {
        OFFSET   0.000000   5.190000   0.000000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET   0.000000   4.140008   0.000000
        }
      }
    }
    JOINT LeftCollar
    {
      OFFSET   1.060000   15.330000   01.760000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftShoulder
      {
        OFFSET   5.00000   0.000000   0.000000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftElbow
        {
          OFFSET   0.000000  -11.90000   0.000000
          CHANNELS 3 Zrotation Xrotation Yrotation
          JOINT LeftWrist
          {
            OFFSET   0.000000  -9.520000   0.000000
            CHANNELS 3 Zrotation Xrotation Yrotation
            End Site
            {
              OFFSET   0.000000  -7.0   0.000000
            }
          }
        }
      }
    }
    JOINT RightCollar
    {
      OFFSET  -1.060000   15.330000   01.760000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightShoulder
      {
        OFFSET  -5.000000   0.000000   0.000000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightElbow
        {
          OFFSET   0.000000  -11.900000   0.000000
          CHANNELS 3 Zrotation Xrotation Yrotation
          JOINT RightWrist
          {
            OFFSET   0.000000  -9.520000   0.000000
            CHANNELS 3 Zrotation Xrotation Yrotation
            End Site
            {
              OFFSET   0.000000  -7.0   0.000000
            }
          }
        }
      }
    }
  }
  JOINT LeftHip
  {
    OFFSET   3.430000   0.000000   0.000000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
    {
      OFFSET   0.000000  -19.0   0.000000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftAnkle
      {
        OFFSET   0.000000  -15.00000   0.000000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET   0.000000  -3.0   0.000000
        }
      }
    }
  }
  JOINT RightHip
  {
    OFFSET  -3.430000   0.000000   0.000000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightKnee
    {
      OFFSET   0.000000  -19.0   0.000000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightAnkle
      {
        OFFSET   0.000000  -15.00000   0.000000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET   0.000000  -3.0   0.000000
        }
      }
    }
  }
}
```

```
MOTION
Frames: 22
Frame Time: 0.033333333

/////// FIRST FRAME
0.000000  28.389458  0.000000  -1.493607  -5.771171  77.969531
  5.046457  3.786636  7.183324  -350.332988  -327.580535
  -3.451620  -2.361056  -27.580998  0.479380  -336.322795
  -335.814820  -17.095797  18.697579  3.325927  14.837262
  120.380278  -136.990054  104.793957  1.610204  -10.989013
  57.487057  357.883570  -335.287147  -5.692479  -49.625715
  -31.107061  61.377717  -77.667363  -31.425819  -44.774467
  -188.048654  -181.790835  98.614494  5.304022  11.097790
  17.950025  -32.184926  103.909536  34.037239  0.825958
  2.017271  14.096251  15.530982  -11.238403  6.982921
  -8.118375  60.475115  7.476830  356.731126  332.946414
  -26.180466

/////// SECOND FRAME
-0.423467  27.926288  0.125110  -4.739621  -7.109743  78.399709
  7.635020  1.345446  8.020145  -351.367597  -328.066708
  -3.014721  357.283743  332.198235  -1.262286  -336.782248
  -335.699909  -15.606101  18.444881  7.242405  15.123302
  118.591233  -140.357050  99.674976  1.894714  -9.179514
  57.405512  355.511637  -335.302938  -5.222891  -49.797112
  -37.572497  59.639762  -84.077646  -23.537892  -44.802703
  -196.640130  -187.281651  97.243641  9.091184  -2.207557
  20.788143  -25.274115  115.581373  26.529762  -0.433875
  1.571555  14.676017  17.057401  -10.096450  7.301430
  -7.346258  60.394088  6.873692  358.320074  329.633335
  -25.335242
...
```

A.2 Drinking Animation

```
HIERARCHY
ROOT Pelvis
{
  OFFSET     0.000000     0.000000     0.000000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation
   Yrotation
  JOINT LowerBack
  {
    OFFSET    -0.000691     0.176488     0.017766
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Chest
    {
      OFFSET    -0.002754     0.128976    -0.000512
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT back3
      {
        OFFSET     0.002471     0.201657    -0.000850
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT Neck
        {
          OFFSET     0.001840     0.072570     0.000595
          CHANNELS 3 Zrotation Xrotation Yrotation
          JOINT Head
          {
            OFFSET     0.003323     0.152818     0.030907
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT joint7
            {
              OFFSET     0.000612     0.183936     0.002625
              CHANNELS 3 Zrotation Xrotation Yrotation
              End Site
              {
                OFFSET     0.000000     0.000000     0.018310
              }
            }
          }
        }
        JOINT LeftCollar
        {
          OFFSET     0.053456     0.093486     0.002205
          CHANNELS 3 Zrotation Xrotation Yrotation
          JOINT LeftShoulder
          {
            OFFSET     0.112884    -0.020868     0.002319
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT LeftElbow
            {
              OFFSET     0.297302     0.003523    -0.020737
              CHANNELS 3 Zrotation Xrotation Yrotation
              JOINT LeftWrist
              {
                OFFSET     0.290007     0.004431    -0.007313
                CHANNELS 3 Zrotation Xrotation Yrotation
                JOINT lf_hand
                {
                  OFFSET     0.117001    -0.002589    -0.008771
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  End Site
                  {
                    OFFSET     0.000000    -0.009821     0.000000
                  }
                }
              }
            }
          }
        }
        JOINT RightCollar
        {
          OFFSET    -0.048666     0.096092     0.000301
          CHANNELS 3 Zrotation Xrotation Yrotation
          JOINT RightShoulder
          {
            OFFSET    -0.115072    -0.022479    -0.001894
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT RightElbow
            {
              OFFSET    -0.297072     0.004954    -0.020752
              CHANNELS 3 Zrotation Xrotation Yrotation
              JOINT RightWrist
              {
                OFFSET    -0.292425    -0.005826    -0.023258
                CHANNELS 3 Zrotation Xrotation Yrotation
                JOINT rt_hand
                {
                  OFFSET    -0.118185    -0.001061    -0.015047
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  End Site
                  {
                    OFFSET     0.000000    -0.009987     0.000000
                  }
                }
              }
            }
          }
        }
      }
      JOINT joint13
```

```
      {
        OFFSET     0.125914     0.117470     0.000000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET     0.000000     0.000000    -0.017144
        }    }
      JOINT joint14
      {
        OFFSET    -0.120063     0.114907     0.000000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET     0.000000     0.000000    -0.016479
        }
      }
    }
  }
  JOINT RightHip
  {
    OFFSET    -0.109283    -0.057257     0.000987
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightKnee
    {
      OFFSET     0.002149    -0.429339     0.038617
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightAnkle
      {
        OFFSET    -0.001765    -0.522522    -0.077092
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT r_ball
        {
          OFFSET    -0.000000    -0.037315     0.142913
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET     0.000000     0.000000     0.014648
          }
        }
      }
    }
  }
  JOINT LeftHip
  {
    OFFSET     0.098180    -0.056757     0.003895
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
    {
      OFFSET     0.007057    -0.430531     0.035812
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftAnkle
      {
        OFFSET    -0.005475    -0.521755    -0.078136
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT lf_ball
        {
          OFFSET     0.000065    -0.036737     0.143337
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET     0.000000     0.000000     0.014648
          }
        }
      }
    }
  }
```

```
}

MOTION
Frames: 1409
Frame Time: 0.033333

/////// FIRST FRAME
0  0  0  -1.833817  -1.835176  -23.948662  -2.461222  -1.578721
  3.752221  0.018913  21.754278  0.865438  0  0  0  0  0  0
  3.728722  -14.91721  -16.28001  0  0  0  14.80264  -1.51327
  -5.718119  -100.91745  -21.587351  20.34293  0.722644
  1.692521  -45.803242  6.67026  7.743932  -3.438003  0  0  0
  -8.552869  -0.641647  4.260422  96.079376  -3.464883
  -4.640043  -0.682071  1.656317  44.761375  -18.083363
  20.402439  -11.308698  0  0  0  0  0  0  0  0  0  1.678106
  6.444203  -23.379509  0  0.811551  0  1.525094  -11.174504
  0.282784  0  0  0  3.804609  -5.665328  -8.418815  0  2.455757
  0  -5.268886  -6.663751  -8.501185  0  0  0

/////// SECOND FRAME
0  0  0  -1.44767  -2.126195  -25.414595  -2.95842  -2.308424
  3.979491  -0.688546  20.408722  1.574033  0  0  0  0  0  0
  3.304029  -13.769586  -16.218054  0  0  0  14.746619
  -1.603417  -6.053154  -99.852943  -20.856672  23.007175
  0.704385  1.684865  -45.373886  4.816781  12.135592
  -0.554579  0  0  0  -8.066298  -0.54358  3.826707  94.945763
  0.074585  -6.30547  -0.644933  1.618572  43.4482
  -11.704623  23.622774  -16.08967  0  0  0  0  0  0  0  0  0
  0.641872  7.57687  -22.781996  0  1.371283  0  2.256754
  -11.28161  1.277745  0  0  0  3.429863  -4.011049  -6.788782
  0  2.812614  0  -6.406186  -7.890936  -7.836373  0  0  0
...
```

Appendix B

Detailed Pseudo-codes for Proposed Techniques

B.1 Curve Fitting with Dynamic Programming

```
#define THRESHOLD_COST 0.01

extern KeyFrameData **rootKey;
extern int totalChannels;
extern KeyFrame **rootResult;
extern int outputtype;

// The following methods are used in the algorithm.
// Their implementations are excluded for the sake
// space considerations but their actions are explained.

void doRotationFix(KeyFrameData **bvhRoot, int chCount)
// This changes certain values in rotation channels. Some
// motion capture data may have rotation data in range
// (0,360) (degrees) and some other data may have rotation
// data in range (-180,180). Thus if an object rotates in
// a direction for more than one full rotation, there will
// be jumps on the values. This method makes sure that
// all rotation information is continious.

void countKeyFrames(KeyFrameData **bvhRoot, int chCount);
// This method simply counts the number of keyframes that
// the algorithm has found.

int EstimateInitials(KeyFrameData **bvhRoot, int chCount);
// This method estimates the initial keyframes. When the
// first and second derivatives of a joint is zero, this
// method places a keyframe at that point.

int DetermineSearchSpaces(KeyFrameData **bvhRoot, int chCount);
// This method finds the initial search spaces for every keyframe.
// It checks the spaces between the keyframes and proposes a
// possible size for the search window.

int DetermineSearchSpaceList(KeyFrameData **bvhRoot, int chCount);
// This method finds all possible values for a keyframe in its
// search window. The algorithm goes over this list.

int DecreaseAllSearchSpace(KeyFrameData **bvhRoot, int chCount);
// This method decreases all of the search spaces to make
// the keyframe search windows ready for the next iteration.

int ConvertToBezierCurve(KeyFrameData **bvhRoot, int chCount);
// This method converts a given segment of a Hermite curve
// definition into a Bezier curve.

int FindBezierValues(KeyFrameData **bvhRoot, int chCount);
```

```
//Given a Bezier curve, this method finds all the
// inner values between the keyframes by using
// Bezier subdivision algorithm.

int CalculateCosts(KeyFrameData **bvhRoot, int chCount);
//This method finds the cost between the simplified curve
// and the original curve. This value is the error value.

int LoadBVHFile(char *filename);
// This method loads a Biovision Hierarchical data file
// which is commonly used for storing motion capture
// data. It saves the information in terms of
// KeyframeData class which the algorithm uses.
// Each KeyframeData object has unlimited number of
// channels and each channel has unlimited number of
// frames. Each frame can store a time and a position/
// rotation value. Each frame can be marked as a keyframe
// and at this point it can store time position/rotation
// value and to find curvature it can store tangent of
// the curve at that keyframe.

int SaveBVHFile(char *baseFilename, char *newFilename,
                KeyFrameData **bvhRoot, int chCount);
// If a new simplified curve is found with keyframes,
// this curve can again be saved as a BVH file, to
// be able to use it on commonly used 3D animation
// softwares.

// The main part
int main()
{
  char infilename[]="old_m_tea.bvh";
  char outfilename[]="old_m_tea_result.bvh";
  char outfilenamenum[40];
  char outfilenamenumtmp[40];
  std::cout << "PROGRAM STARTED" << endl;

  LoadBVHFile(infilename);
  doRotationFix(rootKey,totalChannels);
  EstimateInitials(rootKey, totalChannels);
  DetermineSearchSpaces(rootKey, totalChannels);
  DetermineSearchSpaceList(rootKey, totalChannels);
  ConvertToBezierCurve(rootKey, totalChannels);
  FindBezierValues(rootKey, totalChannels);
  CalculateCosts(rootKey,totalChannels);

  int counter=0;
  while(KeyFrameData::tmpStorageTotalCost>THRESHOLD_COST
   && counter<10)
  {
    ApplyDynamicProgramming(rootKey,totalChannels);
    PathList::clearAll(rootKey,totalChannels);
    DecreaseAllSearchSpace(rootKey,totalChannels);
    DetermineSearchSpaceList(rootKey, totalChannels);

    ConvertToBezierCurve(rootKey, totalChannels);
    FindBezierValues(rootKey, totalChannels);

    CalculateCosts(rootKey,totalChannels);
    std::cout << "COST " <<
    KeyFrameData::tmpStorageTotalCost << std::endl;
    counter++;
  }

  ConvertToBezierCurve(rootKey, totalChannels);
```

```
  FindBezierValues(rootKey, totalChannels);

  countKeyFrames(rootKey,totalChannels);
  std::cout << "Total KeyFrames: " <<
  KeyFrameData::tmpStorageTotalKeyFrames << std::endl;
  std::cout << "Total Frames: " <<
  KeyFrameData::tmpStorageTotalFrames << std::endl;
  std::cout << "Total Channels: " <<
  KeyFrameData::tmpStorageTotalChannels << std::endl;
  std::cout << "Average KeyFrame per channel: " <<
  ((float)KeyFrameData::tmpStorageTotalKeyFrames/
  KeyFrameData::tmpStorageTotalChannels) << std::endl;

  SaveBVHFile(infilename,outfilename,rootKey,totalChannels);
  std::cout << "PROGRAM TERMINATED" << std::endl;

  return 0;
}

int ApplyDynamicProgramming(KeyFrameData **bvhRoot, int chCount)
{
  for (int i=0; i<chCount; i++)
  {
    _applyDynamicProgrammingInChannel(bvhRoot[i]);
    std::cout << "\r dp " << i << "/" <<
    chCount << "                                               ";
    std::cout.flush();
  }
  return 0;
}

void _applyDynamicProgrammingInChannel(KeyFrameData *channelRoot)
{
```

```
KeyFrameData *curData=channelRoot;
KeyFrameData *preData=0;

PathList allPaths;
while (curData!=0)
{
  if (curData->tmpStorage2==1)
  {
    if (preData!=0)
    {
      preData->loadSearch(preData->searchList.root);
      curData->loadSearch(curData->searchList.root);
      while (preData->lastSearch!=0)
      {
        while (curData->lastSearch!=0)
        {
          // do calculation and save result for this case
          //generate the curve and values for each frame
          _convertToBezierCurveInChannel(channelRoot);
          _findBezierValuesInChannel(channelRoot);

          //find the cost for this search
          _calculateCostInChannel(channelRoot);

          PathManager *curPath= curData->
            listPath.findSpecificSearchIndex(curData->searchIndex);
          if (curPath == 0)
          {
            curPath=new PathManager();
            curData->listPath.insertItem(curPath);
            curPath->insertItemHead(curData);
            curPath->insertItemHead(preData);
            curPath->belongsToSearchIndex=curData->searchIndex;
            curPath->minCost=preData->segmentTotalCost;
          }
          else
          {
            if (curPath->minCost>preData->findMinPath()->minCost+
            preData->segmentTotalCost)
            {
              curPath->removeItemHead();
              curPath->insertItemHead(preData);
              curPath->minCost=preData->minPath->minCost +
               preData->segmentTotalCost;
            }
          }
          curData->loadSearch(curData->lastSearch->nextItem);
        }
        preData->loadSearch(preData->lastSearch->nextItem);
      }
    }
    preData=curData;
  }
  curData=curData->nextKey;
}
// end of while loop

curData=channelRoot;
preData=channelRoot;
while (curData!=0)
{
  if (curData->tmpStorage2==1)
  {
    preData=curData;
  }
  curData=curData->nextKey;
  }
  // now predata holds the last keyframe,
  // which is our result

  preData->findMinPath();
  preData->loadReal(preData->minPath->root->nextItem);
      //this is the updating phase

  while (preData!=0)
  {
    if (preData->listPath.root!=0)
    {
      preData->findMinPath();
      preData->prevKey->loadReal(preData->minPath->root);
    }

    do{
      preData=preData->prevKey;
    }while (preData!=0 && preData->tmpStorage2!=1);
  }
}
```

B.2 Curve Simplification

```
//The main part.
void main()
{
  char infilename[]="t1.bvh";
  char outfilename[]="t1_result.bvh";

  std::cout << "PROGRAM STARTED" << endl;
  LoadBVHFile(infilename);
  KeyFrameData::tmpStorageTotalChannels=totalChannels;
  Curve_StartAlgorithm(rootKey,totalChannels);
  curve_countKeyFrames(rootKey,totalChannels);

  std::cout << "Total KeyFrames: " <<
  KeyFrameData::tmpStorageTotalKeyFrames << std::endl;
  std::cout << "Total Frames: " <<
  KeyFrameData::tmpStorageTotalFrames << std::endl;
  std::cout << "Total Channels: " <<
  KeyFrameData::tmpStorageTotalChannels << std::endl;
  std::cout << "Average KeyFrame per channel: " <<
  ((float)KeyFrameData::tmpStorageTotalKeyFrames/
  KeyFrameData::tmpStorageTotalChannels) << std::endl;
  std::cout << "PROGRAM TERMINATED" << std::endl;
}

// This method counts the number of keyframes in a channel
void _curve_countKeyFramesInChannel
         (KeyFrameData *channelRoot)
{
  KeyFrameData *curData=channelRoot;
  KeyFrameData *lastData=0;
  while (curData!=0)
  {
    if (curData->curve_isKey==1)
      KeyFrameData::tmpStorageTotalKeyFrames++;
    KeyFrameData::tmpStorageTotalFrames++;
    curData=curData->nextKey;
  }
}

// This method uses _curve_countKeyFramesInChannel
// and counts all the keyframes
```

```
void curve_countKeyFrames
        (KeyFrameData **bvhRoot, int chCount)
{
  KeyFrameData::tmpStorageTotalKeyFrames=0;
  KeyFrameData::tmpStorageTotalFrames=0;
  for (int i=3; i<chCount; i++)
  {
    _curve_countKeyFramesInChannel(bvhRoot[i]);
  }
}



// This method finds the error ratio between
// a beginning point, an ending point and a
// middle point. The more three points are
// closer to a line, the lower the error ratio.
// It calculates the area of a triangle since
// it knows the lengths of the sides, then uses
// the base side length (from beginning to the
// end point) to find the height of the triangle.
// This is the error value, it uses this height
// value with the base value to find a ratio.

float findErrorDistanceRatio(float bx, float by,
          float cx, float cy, float ex, float ey)
{
  float a=sqrt(pow(cy-by,2)+pow(cx-bx,2));
  float b=sqrt(pow(ey-by,2)+pow(ex-bx,2));
  float c=sqrt(pow(cy-ey,2)+pow(cx-ex,2));

  float s=(a+b+c)/2.0f;
  float area=sqrt(s*(s-a)*(s-b)*(s-c));
  float h=area/b;
  return h/b;
}

// This function implements the recursive
// curve simplification algorithm

void Curve_DoRecursive
       (KeyFrameData *first, KeyFrameData *last)
{
  KeyFrameData *curData=first;

  float maxError=0;
  KeyFrameData *maxErrorNode=0;

  float curError=0;
  while (curData!=last)
  {
    curError=findErrorDistanceRatio(
    (float)first->keyTime,first->keyValue,
    (float)curData->keyTime,curData->keyValue,
    (float)last->keyTime,last->keyValue);
    if (curError>maxError)
    {
      maxError=curError;
      maxErrorNode=curData;
    }

    curData=curData->nextKey;
  }

  if (maxError>0.005f)
  {
    std::cout << "\r\tNew Keyframe at " <<
     maxErrorNode->keyTime << "                    ";
    maxErrorNode->curve_isKey=1;
    Curve_DoRecursive(first,maxErrorNode);
    Curve_DoRecursive(maxErrorNode,last);
  }
}

// This method makes sure that the curve
// simplification algorithm runs on a channel.

void _curve_StartAlgorithm_inchannel(KeyFrameData *r)
{
  KeyFrameData *tmp1=r;
  KeyFrameData *tmp2=r;
  while (tmp2->nextKey!=0)
  {
    tmp2=tmp2->nextKey;
  }
  tmp1->curve_isKey=1;
  tmp2->curve_isKey=1;
  Curve_DoRecursive(tmp1,tmp2);
}

// This function starts the curve simplification
// algorithm on every channel

void Curve_StartAlgorithm
        (KeyFrameData **r, int totalChannels)
{
  for (int i=3; i<totalChannels; i++)
  {
    std::cout << "Curve simplification algorithm in channel " <<
    i << " of " << totalChannels << "\n";
    _curve_StartAlgorithm_inchannel(r[i]);
  }
}
```