Motion Segmentation Based On Joint Swings

Chia Wen Jie Alvin

Department of Computer Science

School of Computing

National University of Singapore

2009

Abstract

Synthesizing new motion is a difficult problem. Synthesis through physical simulation produces the best results but it suffers from the amount of time needed and thus, it is not suitable for real time use such as in a game. Therefore, an approach of synthesis using existing motion would be more suited for real time application. However, creating new motion from existing ones is a challenging task because of the motion data generally lacks structure and intuitive interpretation. We have come out with a novel motion segmentation model based on the dynamics of the motion which enables us to modify the intensity and timing of existing motion. For example, we could make a kick much more forceful, or change the duration of the kick. We believe our model could be used for motion compression as well as help in motion analysis in general because it encodes temporal, spatial and intensity information.

Subject Descriptors:

I.3.6 Computer Graphics Methodology and Techniques

I.3.7 Three-Dimensional Graphics and Realism

Keywords:

Character Animation, Algorithms, Data Structures, Computer Graphics

Implemented Software and Hardware:

Microsoft Visual Studio 2008 C++, OpenGL, Microsoft Foundation Class (MFC)

Acknowledgment

I would like to thank my supervisor Prof Leow Wee Kheng for providing me guidance and support for this work.

This project will also not be possible without the help of friends and mentors who have always given me valuable feedbacks along the way and I would like to take this opportunity to thank them.

Table of Content

Chapter 1
Introduction

1.1 Motivation

Motion Synthesis is the creation of motion data. It can be done generally in 2 ways: either through physical simulation or by creating new motion from existing motion (Exemplar Based). Motion Synthesis is usually done to create new motions for animations, videos games as well as virtual environment such as 2nd Life. Such applications do not only require large amount of motions for the characters, variety of motions is also very important. In this thesis, when we talk about motion synthesis, we meant the exemplar based approach and not the physical simulation way.

Such animations are normally done in 2 ways: requiring skilled animators who manually hand animate 3D characters in software packages like Maya and 3DS Max or using motion capture, where actors wearing special suits “acted” out the required motion which are then captured and stored. Manual animation requires skilled animators and is very time consuming while motion capture is expensive and the resultant motion might not meet the requirements because of noise or simply because the timing of the actor is off.

This is where motion synthesis comes in, it allows the creation of new motions which when done right, can satisfy the requirements of the application. Furthermore, it allows reuse of existing motion data, which would be wasted since normally they are used in an application once and discarded because it is very hard for another application to use it without having to editing or modify it.

Motion synthesis is usually done by manipulating the motion data in its raw form. Figure 1 shows a plot of all the joint angles in a motion against time. It is hard to decipher what the motion is doing by only looking at the plots.

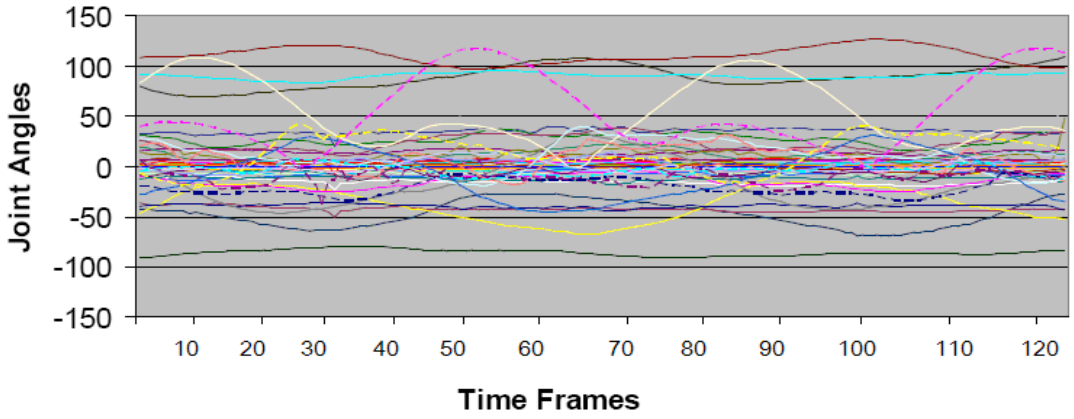

Figure 1: Plot of all the joint angles against time. Note how difficult it is so know what the motion is doing.

However, because the raw motion data itself is unstructured, it not trivial to get information such as the swinging of the joints and how fast it is swinging.

Therefore, to derive meaning from motion data, one way would be to build a hierarchical model to on top of the motion data. A hierarchical model is a way to derive meaning from multimedia signals such as video, audio and in our case, motion data. Taking audio, in particular speech as an example, we can analyze to break down speech into phonemes and then combine phonemes into syllables and finally combine syllables into words. To break down the audio signal into these components, we need to perform segmentation on the audio signal to know the start and end of the phonemes. As far as we know, no one has done this before for motion data. Similar to how it is done for video and audio clips, building a hierarchical model requires that segmentation be done. To do this, we need a segmentation model and this leads us to the objective of this thesis.

1.2 Thesis Objective and Contribution

The objective of this thesis is to develop a segmentation model for motion data to model the swings of the motion. For instance, when the arm is swinging forward during a run, the humerus (the bone where the bicep and tricep are) is swinging forward in a rather geodesic manner. For example, in the picture below, the right upper arm will swing forward and back as the character runs

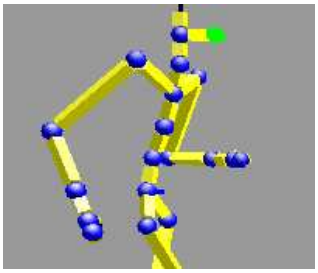

Figure 2: The right arm will swing forward during a run.

This means, the bone is rotating around a fairly constant axis of rotation when swinging forward. This applies to the other bones such as those on the legs as well. Thus, we will have taken a step forward in the building of a hierarchical model for motion data if we can build this segmentation model.

Such swings can be segmented from the motion data and it's analogy with respect to video would be shots in shot detection.

Therefore, the main objective of this thesis is to come out with a segmentation model which will be able to segment out these swings. If we playback just these swings instead of the original motion data, we should get a good approximation of the original motion. This would demonstrate that the model does indeed work.

With these swings segmented out, we can show that we can do use them to some applications, namely:

- A fairly basic form of motion compression by just storing these swings
- A way to index and search for motion in a motion database by using these the segmentation result
- Simple motion editing by manipulating the properties of these swings

1.3 Thesis Organization

The rest of the thesis will be organized as such. We will talk about some Background Knowledge about animating a skeleton with motion data because this is a very basic

requirement in dealing with motion data. Then we go on to Related Work where we discuss some relevant works in the literature. After that would be discussion on our Motion Segmentation Model and how it is done. Following that would be on how we can make use of the segmentation model and we end with the conclusion.

Chapter 2

Background Knowledge

2.1 Animating a Skeleton using Motion Data

2.1.1 The Skeleton Structure

The skeleton is a structure similar to our human skeleton. It is made up of bones and joints. The bones are typically named according to their biological name. So for example, the thigh bone is called the femur.

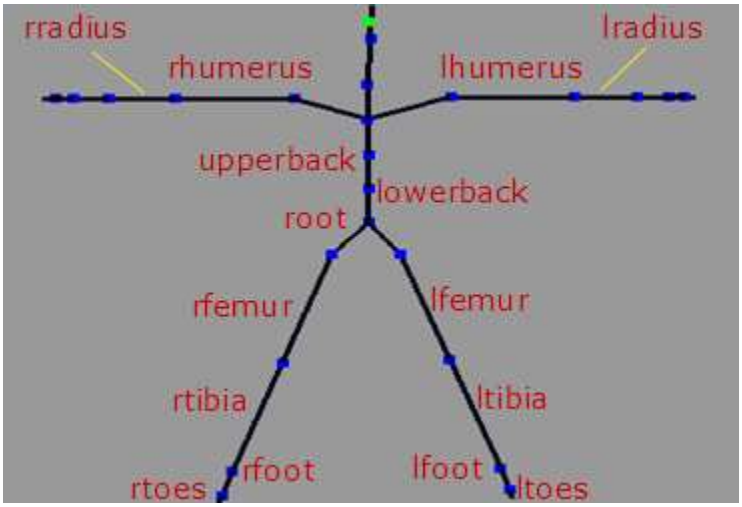

Figure 3: An example skeleton and the bone names. The labels give the medical name for each bone.

The skeleton structure is actually a tree with the root joint as the root of the tree. So, the child of the lfemur is the ltibia. Each child will store the transformation information from its parent to itself. The position of the skeleton in Figure 2 is known as the bind-pose.

2.1.2 Animating the Skeleton

How do we animate the skeletal structure? We do so by specifying the rotation of each individual bone with respect to the parent bone's local coordinate frame. Different bones have varying number of Degrees of Freedom (DOFs). For example, the femur (thigh) can rotate freely in the x, y and z axis while the radius (fore-arm) can only rotate in one axis. Typically, depending on the motion capture equipment used, we will know the order to apply the x, y and z rotation for each bone/joint.

The root is the only bone with translation component to it and whose rotation component is with respect to the world coordinate frame. Therefore, it has 6 DOFs. Any translation of the skeleton in 3D is specified by the translation component of the root.

The structure of the bones as well as the sequence of angles for each bones form what we know as the motion data.

A collection of the angles for each bone specifies a *pose* for the skeleton. We call such a collection a frame. A sequence of frames would give us the animation of the skeleton performing whatever motion is captured. Motion capture is usually captured at 120 frames per second and then down scaled to 60 frames per second. This is what is done with the motion capture data from the CMU Graphics Lab. Therefore, each frame represents 1 / 60 of a second, and from the number of frames in a motion, we can work out the duration of the motion.

As mentioned, the pose in Figure 3 is known as the bind-pose and it is usually specified by having all the DOF of each bone set to 0.

2.1.3 Animation a 3D mesh

To drive a 3D mesh of a human using the skeleton, we have to perform a process known as skinning. Given a 3D mesh and a skeleton,

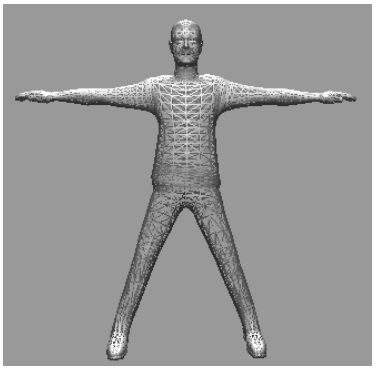

Figure 4: 3D Mesh. A sample 3D human mesh.

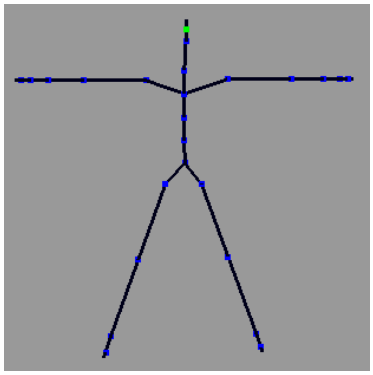

Figure 5: A Skeleton.

A simple skinning method is to assign 1 bone to each vertex. There is more sophisticated method to do skinning but for our purposes, this simple method is enough to illustrate the process. The purpose of assigning a bone to each vertex is so that when the bone is rotated as specified by the motion data, the vertices "follow" it, thus creating an animation.

Before we can assign a bone, we must position the skeleton "inside" the mesh so that the when the bone rotates, the vertices that follows will do so correctly. The image below shows the skeleton inside the mesh.

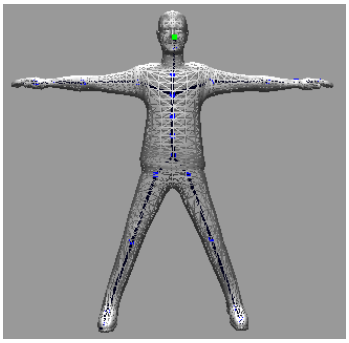

Figure 6: Skeleton positioned inside the mesh.

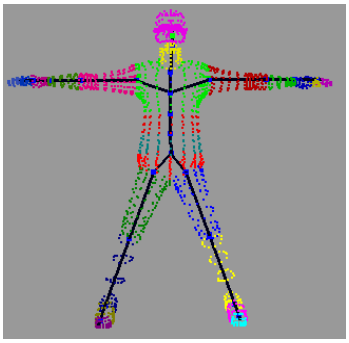

Figure 7: Color coding the vertex assignment.

Once skinning is done, the character is said to be "rigged" and is ready to be animated.

2.2 Representing Rotations

There are a number of ways to represent rotations in 3D.
Below are some of them

- Euler Angle [refer to citation 18 for more details]
- Rotation Matrix [refer to citation 19 for more details]
- Quaternion [refer to citation 20 for more details]
- Exponential Map [refer to citation 17 for more details]

Euler Angle representation is the most straight forward way where the rotation is represented by a 3x1 vector corresponding to rotation with respect to x, y and z axes respectively. It is developed by Leonhard Euler to describe the orientation of a rigid body (a body in which the relative position of all its points is constant) in 3D Euclidean Space. However, one well known problem with Euler Angle representation is that it is plagued by the gimbal lock problem.

Gimbal lock is the loss of one degree of freedom that occurs when the 1 of the 3 axes (in a 3D space) becomes aligned with one of the remaining 2 axes. This results in a loss of the degree of freedom for the particular axis. Refer to [21] for a more in depth explanation.

We can use the rotation matrix representation, where each rotation is encoded by a 3x3 matrix. This does not suffer from the gimbal lock problem. However, a 3D rotation is has only 3 degrees of freedom, namely the angle to rotate for each principal axis but the rotation matrix has 9 components. This is not suitable for applications where memory constraints apply.

There is also the quaternion representation where a rotation is represented compactly by a 4x1 vector. Rotation can be performed in quaternion space and it does not suffer from gimbal lock as well. However, quaternion have a strict rule to be of unit length, otherwise, the 4x1 vector representing the quaternion would not make any sense. This makes

quaternion unsuitable for applications where interpolations and making small changes to rotations are required because the quaternion has to be renormalized each time it is changed.

This is where exponential maps come in. Exponential maps attempts to map a 3D rotation to a vector in R^3 space by having the R^3 vector represent the axis of rotation and the magnitude of the vector specifying the angle to rotate using the Right Hand Rule. This is not possible without the possibility of gimbal lock. However, the gimbal lock in exponential map is avoidable and that makes it suitable as a replacement for quaternion. We can convert from Exponential Map to Quaternion as shown below.

$$e^{[0,0,0]^T} = [0,0,0,1]^T \qquad \text{and for } \mathbf{v} \neq \mathbf{0} \qquad e^{\mathbf{v}} = \left[\sin(\tfrac{1}{2}\theta)\hat{\mathbf{v}},\ \cos(\tfrac{1}{2}\theta)\right]^T$$

where $\theta = |\mathbf{v}|$ and $\hat{\mathbf{v}} = \mathbf{v}/|\mathbf{v}|$.

The only problem with this particular formulation is that calculating $\hat{\mathbf{v}} = \mathbf{v} / |\mathbf{v}|$ as $\mathbf{v}$ goes to zero becomes numerically unstable. However, by rearranging the above formula a bit, we will be able to see that this exponential map *can* be computed robustly even in the neighborhood of the origin.

Let

$$\mathbf{q} = e^{\mathbf{v}} = \left[\sin(\tfrac{1}{2}\theta)\frac{\mathbf{v}}{\theta},\ \cos(\tfrac{1}{2}\theta)\right]^T = \left[\frac{\sin(\frac{1}{2}\theta)}{\theta}\mathbf{v},\ \cos(\tfrac{1}{2}\theta)\right]^T$$

All we have done is reorganize the problematic term so that instead of computing $\boldsymbol{v} / |\boldsymbol{v}|$ (*i.e.* $\boldsymbol{v}/\theta$), we compute $\sin(\frac{1}{2}\theta) / \theta$. This is because $\sin(\frac{1}{2}\theta)/\theta = \frac{1}{2}\mathrm{sinc}(\frac{1}{2}\theta)$, and sinc is a function that is known to be computable and continuous at and around zero. Assured that the function *is* computable, we still need a formula for computing it, since sinc is not included in standard math libraries. Using the Taylor Expansion of sine function, we get:

$$\frac{\sin(\frac{1}{2}\theta)}{\theta} = \frac{1}{\theta}\,\text{TaylorExpansion}(\sin(\tfrac{1}{2}\theta))$$
$$= \frac{1}{\theta}\left(\frac{\theta}{2} + \frac{\left(\frac{\theta}{2}\right)^3}{3!} - \frac{\left(\frac{\theta}{2}\right)^5}{5!} + \cdots\right)$$
$$= \frac{1}{2} + \frac{\theta^2}{48} - \frac{\theta^4}{2^5 \cdot 5!} + \cdots$$

From this we see that the term *is* well defined, and that evaluating the entire infinite series would give us the exact value. But as $\theta \to 0$, each successive term is smaller than the last, and terms are alternately added and subtracted, so if we approximate the true value by the first *n* terms, the error will be no greater than the magnitude of the $(n+1)^{th}$ term.

The principal advantage of quaternion over Euler angle is their freedom from gimbal lock. We already know that the exponential map must suffers from gimbal lock too, so if it is to be useful, we must know how and where gimbal lock occurs and show how they can be avoided at a cost that is outweighed by its benefits.

The problems with exponential map shows up on the spheres (in R^3) of radius $2n\pi$ (for n=1,2,3,…). This makes sense, since a rotation of 2π about *any* axis is equivalent to no rotation at all – the entire shell of points 2π distant from the origin (and 4π, and so on) collapses to the identity in SO(3) . So if we can restrict our parameterization to the inside of the ball of radius 2π, we will avoid the gimbal lock. Fortunately, each member of SO(3) (except the rotation of zero radians) has two possible representations within this ball: as a rotation of q radians about **v**, and as a rotation of $2\pi - \theta$ radians about **−v**.

By moving through time in small steps (making small changes to the rotation, keeping the change to $< \pi$), we can easily keep orientations inside the ball by doing this: at each time step when the rotation is queried for its value, we examine $|\mathbf{v}|$, and if it is close to π, we replace **v** by $(1-2\pi/|\mathbf{v}|)\mathbf{v}$, which is an equivalent rotation. Such *reparameterization* could be done to the Euler Angles as well, but it is simpler when performed on Exponential Maps since it involves just scaling a 3x1 vector since the magnitude of the

vector represents the angle of rotation. For Euler Angle, a series of trigonometric functions are involved to do this and obviously is more computationally intensive compared to doing the same thing for Exponential Map.

One disadvantage of Exponential Map when compared to quaternion is that there is no simple way to combine rotations. We have to convert the Exponential Map to quaternion, perform quaternion multiplication, and then transform the result back to Exponential Map.

Therefore, in our segmentation model, we make use of Exponential Map to represent the rotations because we need to perform interpolation, averaging and smoothing on sequence of rotations. It is easier to compute using Exponential Map compared to Quaternion.

Chapter 3
Related Works

3.1 Motion Segmentation

There are actually very few works on motion data segmentation. Of those that are around, most of them either require manual tagging (for example, by tagging frames where the feet are supposed to be on the ground) or the data is segmented by finding start and end of a motion. This method is tedious and error prone. As far as we know, there is no segmentation done to find out where the swings of a motion are. Extraction of such low level features has not been done and we believed that with the extraction of such swings might be more useful then determining start and end of motion.

For videos, there are shot detection to detect when a shot starts and ends. A shot is defined as the time when the shoot button is pressed on the recorder to the time when the stop button is pressed. There are many literatures on this and when comparing motion data with video, this is the closest analogy.

In audio, in particular speech processing, speech is model by phonemes and syllabuses, before these are combined into words, phrases etc. We believed, that this can be by extracting swings from motion data, we can do something similar to speech processing, by combining swings into actions such as kick, punch etc.

3.2 Motion Synthesis

There are actually a number of ways to synthesize motion. One way uses physical simulation to simulation the physics of the required motion so as to come out with new ones. Note that physical simulation need not use any existing motion to generate new motions. The other is to use existing motion. We termed it Exemplar-based techniques. We could store all the motions in a database and generate or synthesize new motions by finding suitable example in the database. This is synthesis through multiple examples.

In both ways, synthesis is done through a direct manipulation/handling of the motion data itself.

Physical Simulation, as the name suggests, tries to generate motion by simulating the physics of what would happened given the required conditions. For example, we can specify that a character needs to kick a certain object in space, and given a physically correct model, we can run a simulation to produce the desired motion.

However, Physical Simulation often involves some form of optimization and therefore, it is very slow and is not very suitable for real-time usage such as in games or virtual environment. We will focus on Exemplar-based synthesis techniques instead as mentioned earlier in the 1st chapter of this thesis.

Exemplar-based techniques fall into several categories. Very broadly, these are:

- Concatenation approach
- Time Warping approach
- Signal Processing approach

For Exemplar-based techniques, we can synthesize new motions from many examples, or from a single example. The concatenation approach usually uses many examples to synthesize new ones while the Time Warping and Signal Processing are mainly dealing with 1 motion. We can also say that the 2 approaches are more like Motion Editing rather

than Motion Synthesis. However, I would still label them as Motion Synthesis because the generated motion is “new”; it is not produced by motion capturing.

Note that what the techniques above only synthesize new motion data by dealing with the motion data itself. However we will see that there is one example of editing the skeleton structure and generating new motion data for the new structure.

Our approach falls only in the editing motion data category. We do not modify the skeleton structure at all.

We must mention that for all Exemplar-based motion synthesis, there is almost always post processing being performed after the motion is generated. One common problem is the foot skate problem where the foot “slides” along the ground. However, we can use motion generated from Exemplar-based methods as a “starting point” for animators working on computer games as well as computer generated movies. It would definitely be faster than having the animator create an animation from scratch.

3.2.1 Concatenation Approach

In this approach, the main idea for synthesizing new motion is to take example motion and concatenate them together. The hard part is finding the right place to join different motion sequences together so that the resultant motion is correct.

One method of doing this is to use a Motion Graph. There are actually a number of papers published on the topic of Motion Graph. Motion Graph (Kovar, Gleicher, Pighin, 2002), Interactive Motion Generation from Example (Arikan, Forsyth, 2003) all talks about motion graphs.

The general idea about motion graphs is that the edges are all motion clips while the nodes are the transition points. Consider Figure 7 below which shows a very simple motion graph made of 2 different motion clips.

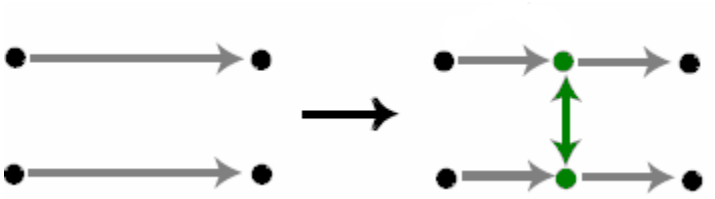

Figure 8: Picture showing example of motion graph.

The 2 horizontal lines on the left are 2 different motion clips. The motion starts on the left and plays to the right. On the right, the green dot and line represents the transition point. So if we start at the top motion, and we play the animation, when we reach the frame at the green dot, we can either choose to continue playing the original motion, or move to the bottom motion and continue the animation from there. Therefore, any walks from the motion graph will be a new sequence of motion.

The challenge then is to locate the transition points. To do this, there need to be a way to compare the between 2 different poses to determine their similarity. Once we have this, we can then come out with a similarity image between 2 different motions. It is generated by comparing each frame in one motion with every frame in the other. The images below

show example of such similarity images. A high similarity will show up as white in the image while low similarity with be darker.

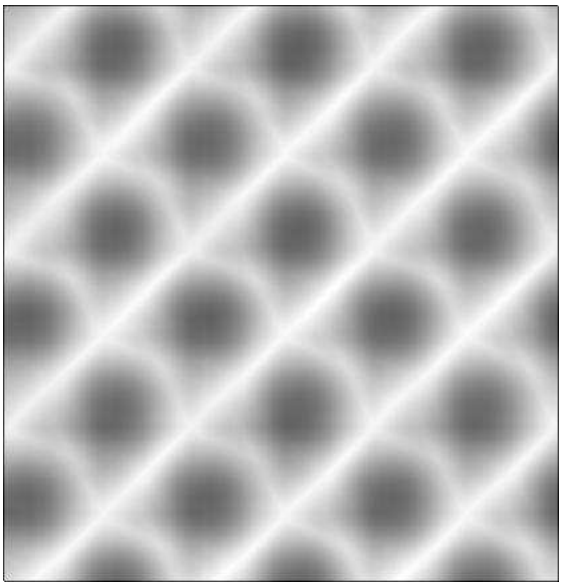

Figure 9: Similarity image between 2 walk motion, note the repeating patterns due to the cyclic nature of walking.

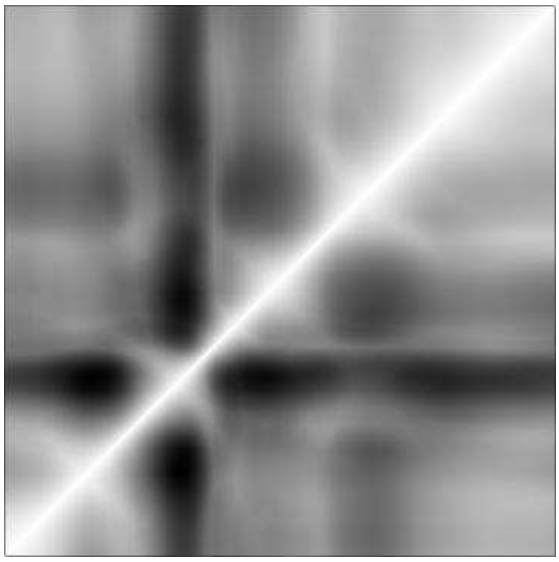

Figure 10: Similarity image for a motion against itself. Note the white diagonal. This is due to comparing the pose in a frame against itself.

Once the similarity images are generated, the transition points can be determined by finding the pair of frames where similarity is high. However, because there is no way we can have perfect matches of poses between 2 different motions, some form of blending must be performed during the transition from on motion to the next.

3.2.2 Time Warping Approach

Time warping is one technique which allows user to adjust the timing of animated characters without affecting their poses. For example, we can adjust a punching motion such that a punch takes longer to execute.

The importance of timing in animation is highlighted in John Lasseter's "Principles of Traditional Animation Applied to 3D Computer Animation" and he notes how even the slightest timing difference can greatly affect the perception of the animation. However, the time warping too requires great skill and patience in order to achieve good result.

Linear time warping is usually used because it is easier to perform. However, recent works have used non-linear time warping which can produce better results.

In the paper, "Guided Time warping for Motion Editing", the author is able to change the timing of a motion by doing non-linear time warping.

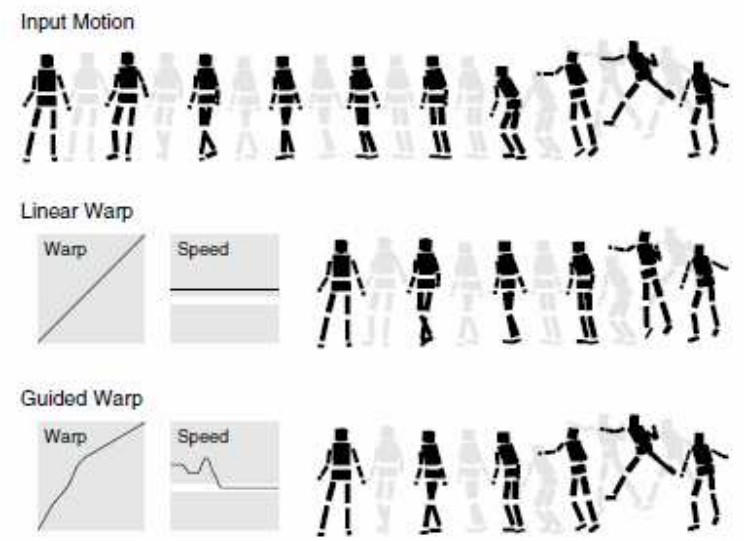

Figure 11: Difference between linear and non-linear time warping. Image taken from the paper

The method generates a retimed output motion based on 2 motions, an input motion which is to be retimed, and a reference motion which controls affects how the input motion is retimed. The output will be similar to the input while matching the "speed" of the reference.

3.2.3 Signal Processing Approach

In this approach, the motion data is treated as a motion signal. The sequence of angles for each DOF of each joint becomes the signal. The signal can then be converted to the frequency domain and the motion can be edited by filtering out unwanted frequencies and converting the signal back to the time domain.

In the paper “Motion Signal Processing”, this is what the authors did. They found out that the main movements in a motion, such as the swinging of the thighs and arms during walking were mainly composed of the lower frequency components. The high frequency components were either noise, or details such as waving of hands.

In the recent SIGGRAPH paper, “Cartoon Animation Filter”, the authors came out with a filter that could easily add anticipation, follow-through and squash-and-stretch to a motion.

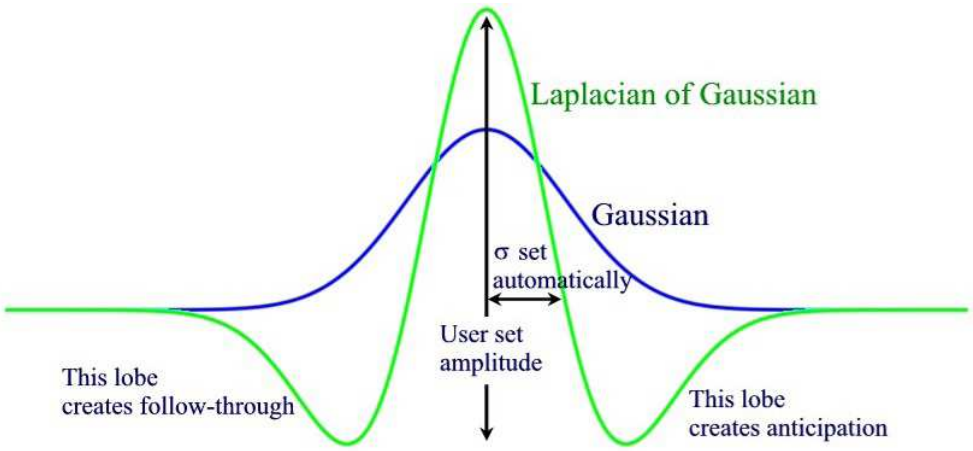

Figure 12: The plot of the cartoon animation filter. Image taken from the paper

The filter is actually a very simple one. It is just an inverted Laplacian of Gaussian. The new motion is obtained by adding a filtered version of the motion signal to itself. The result is quite elegant in that one filter is all that is needed, and there is only one parameter for the user to control. The others can be determined automatically. Therefore, this method can provide a quick and easy way quick come out with new motions from existing ones.

3.2.4 Skeleton Structure Modifying Approach

Modifying the skeleton structure to synthesis new motion seems rather counter intuitive at first. Why modify the skeleton? The skeleton is driven by the motion data so we should focus on manipulating the motion data instead.

On closer examination, we could modify the skeleton structure to generate come up with physically impossible motion is a rather novel idea. The most prominent example is the paper “Rubber-like Exaggeration for Character Animation”, which breaks each bone in the skeleton into smaller ones so as to be able to simulate the “rubbery” effect in cartoons.

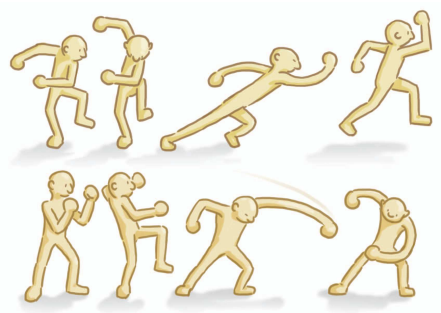

Figure 13: Example of rubber like motion. Note the stretching of the limbs of the character. Image taken from the paper.

Each bone is broken down into several small bones covering the whole length of the original ones as show in Figure 13. For squash and stretch effects, the length of the bones can be “lengthen” during the stretching portion of the animation. Such motion is impossible to be performed by a human being because the human bone can lengthened or shorten at will.

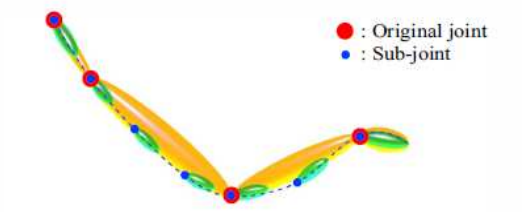

Figure 14: Breaking down of a bone into several smaller ones. Orange represents the original joint while the green ones are the smaller joints used to represent the original ones. Image taken from the paper

Chapter 3
Motion Segmentation

4.1 The Motion Segmentation Model

In this section, we will be discussing the details of our segmentation model as well as the some applications where it could be used.

Before that, let us recall that for each bone/joint in a skeleton, it is driven by a series of rotations applied to it. Each rotation can be described by a rotation matrix. The rotation matrix has only 3 degrees of freedom, which is the angle to rotate in the x, y and z axis respectively. Therefore, the skeleton has a sequence of rotation matrix for each bone/joint describing its pose at a particular frame.

The main idea behind our Segmentation Model is that for highly dynamic motions such as running, kicking or any sports motion, the quick and forceful swinging of the limbs can be parameterized and/or approximated by a rotation axis and a start and end angle. The “rest” period between 2 consecutive swings are usually quite stationary. There might be some movement about but, for most parts, during a highly dynamic movement, the rest period is pretty stationary.

The reason we can do this is that for such forceful swings, the bone/joint going through the motion follows a nearly geodesic path that very nearly rotates around a fixed axis. We can visualize it imaging the centre of rotation of the bone as the center of a sphere with radius equals to the length of the bone. The tip of the bone will then trace a path on the surface of the sphere as it rotates.

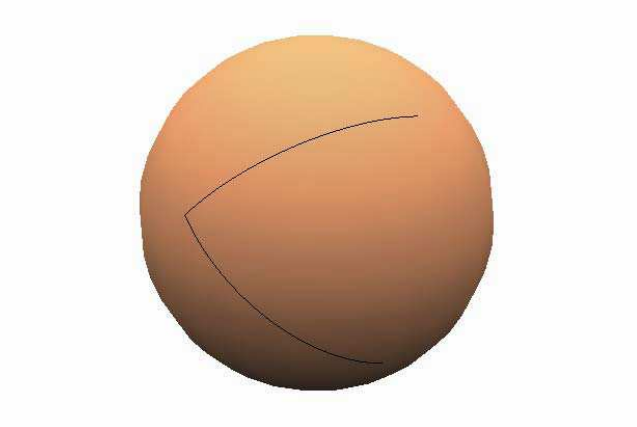

Figure 15: 2 geodesic paths on the surface of a surface. The path shows what a bone might trace through if it were to go through one a perfect geodesic swing.

The image above shows 2 geodesic paths on the surface of a sphere. In highly dynamic motions, the path each limb traces through is highly geodesic. It would not be perfectly geodesic, but it is very close. Our method tries to find out where all these swings are and therefore, by locating all these highly geodesic swings in the motion of every bone/joint, we can segment the motion data by these swings.

Therefore, for each bone/joint, a segment is defined by as the time of the start of a geodesic swing to the time when it ends. Each segment will encode the approximate axis of rotation as well as the start and end angle to rotate.

To show that the segmentation is a good approximation of the original motion, all we have to do is to playback the segmented result and if it the resulting animation is a good representation of the motion, then we will have succeeded.

4.2 Motion Segmentation Algorithm

The rotations of the bones are stored relative to the bone's parent; therefore, all the rotations are local rotations. Using local rotations instead of world rotations makes sense because we are looking for swings for each joint (which is local to each bone). Using world rotation might would not give us consistent results because a swing might no be detected as a swing when using world rotation.

Let $\mathbf{R_{L,j,t}}$ be the local rotation matrix of joint ***j*** at time ***t***. The local rotation matrix for each joint at a particular frame can be obtained from the Euler angles stored in the motion data.

We then obtain the angular velocity of joint ***j*** at time ***t*** from the local rotation matrix. The intuition of velocity is the difference of rotation at time ***t*** and time ***t+1***. Let $\mathbf{\Omega_{j,t}}$ be the angular velocity for joint ***j*** at time ***t***. With respect to rotation, this will be the difference in rotation computed as shown below:

$$\Omega_{j,t} = \begin{cases} R_{L,j,t}^{-1} * R_{L,j,t+1} & t < \#\,of\ frames \\ R_{L,j,t-1}^{-1} * R_{L,j,t}^{-1} & t = \#\,of\ frames \end{cases}$$

The reason for obtaining the angular velocity is because we need to find a way to isolate the individual swings out from the motion data. From the angular velocity, we then obtain an Exponential Map representation. The reason for using Exponential Map is detailed in Chap 2: Background Knowledge.

In simple terms, Exponential Map is a 3 by 1 vector with represents the rotation axis and the magnitude of the vector represents the angle of rotation around the axis. We choose Exponential Map over quaternion representation because it is more stable then quaternion when doing convolution on a sequence of such variables and because its representation is rather simple, the computation is faster compared to quaternion. Let $\mathbf{EXP_{j,t}}$ be the exponential map representation for joint ***j*** at time ***t***.

We then smooth the velocity curve by convolving the exponential map representation with a box filter of with a maximum size of 20 frames. This is to reduce the noise in the original velocity curve and to aid in isolating the geodesic swing.

Through empirical experimentation, we found that most swings have lengths of around 15 to 25 frames. Therefore, the size of the kernel for smoothing the velocity curve is set to be 20 so that the smoothing does not smooth out unnecessary details

Let $\mathbf{EXP_{C,j,t}}$ be the smoothed velocity curve with ***j*** and ***t*** having the same meaning. The next step would be to determine the error of a sequence of 20 frames centered on every frame in the motion data from a perfect geodesic swing with the centered frame's axis of rotation as the axis of rotation for the perfect geodesic swing. Again here, we chose 20 frames because the length of a typical swing is around this number.

We define the error, $\mathbf{E_{j,t}}$ as follows:

$$E_{j,t} = \frac{1}{20} \sum_{i=t-10}^{t+10} \left[\left[\cos^{-1}\left(\frac{EXP_{C,j,i} \bullet EXP_{C,j,t}}{\| EXP_{C,j,i} \| * \| EXP_{C,j,t} \|} \right) \right] * \left[\frac{1}{1+((i-t)/20)^2} \right] \right]$$

The search for the swings can then be performed using the error that we calculated. We will adopt a greedy approach when searching; we start with the frame which has the least error. This is because the swings tend to be centered by the frame with the least error as given above.

Let its frame number be ***f***. This frame will be the starting point of the search for a swing and the convolved angular velocity, $\mathbf{EXP_{C,j,f}}$, at this frame is used as the reference when determining whether to include other frames as part of the swing. We compute the difference of the angular velocity to the left and right of the starting frame, given by the following formula. Let n be the frame number of the frame of either the left or right frame:

$$\Delta_{j,f,n} = -1 * \left(\left[\frac{EXP_{C,j,n} \bullet EXP_{C,j,f}}{\| EXP_{C,j,n} \| * \| EXP_{C,j,f} \|} \right] - 1 \right)$$

We are basically just computing the cosine of the angle between the rotation axis between the 2 frames. Since the cosine function range from 1 to -1, and when 2 frames are on the same axis, the angle between them is 0 and the cosine is 1, we scale the error function accordingly so that the error range from 0 to 2, where 0 indicates perfect match while 2 means the worst match. We add the left or right frame to the original frame depending on which has the smallest error. After that, we average the error in the included frames.

The left or right frame is added into the swing until the average error of the swing exceeds a threshold that we set by ourselves. This threshold controls the quality of swings detected. We call it the **Acceptable Swing Error**. A higher threshold gives us longer swings but they may not be accurate. A smaller threshold gives us shorter swings but they are more accurate. So the threshold could be used as a Level of Detail parameter to control the accuracy of swings extracted. More details about the **Acceptable Swing Error** we used can be found under the results section.

Once we have determined the start and end of a swing, the swing is labeled and attributes such as the axis of rotation, duration, start and end angle. The axis of rotation is obtained by averaging the axis of rotation for each frame in the swing. The search for the next swing will then proceed until all frames have been processed.

For all those frames in between, we will represent by the duration (start and end time) and mean position.

In our work, we only perform this segmentation on the major bone/joint. We exclude bone/joint such as fingers and toes. The reason for this is that the movements of these joints are often too small and may not be enough to be segmented. We tried segmenting them but the results we obtained were not very accurate at all. The motion data on these joints are often very noisy due to the limited resolution of the motion capture equipment used and because of their limited range of movement. Also, we are concentrating on big,

major movement of joints such as the swinging of arms and legs which are mainly responsible for the motion instead of those of the fingers and toes.

4.3 Results

An example of the segmentation result is shown for a running motion below. Therefore, we can say that this result represents one step above the raw motion data.

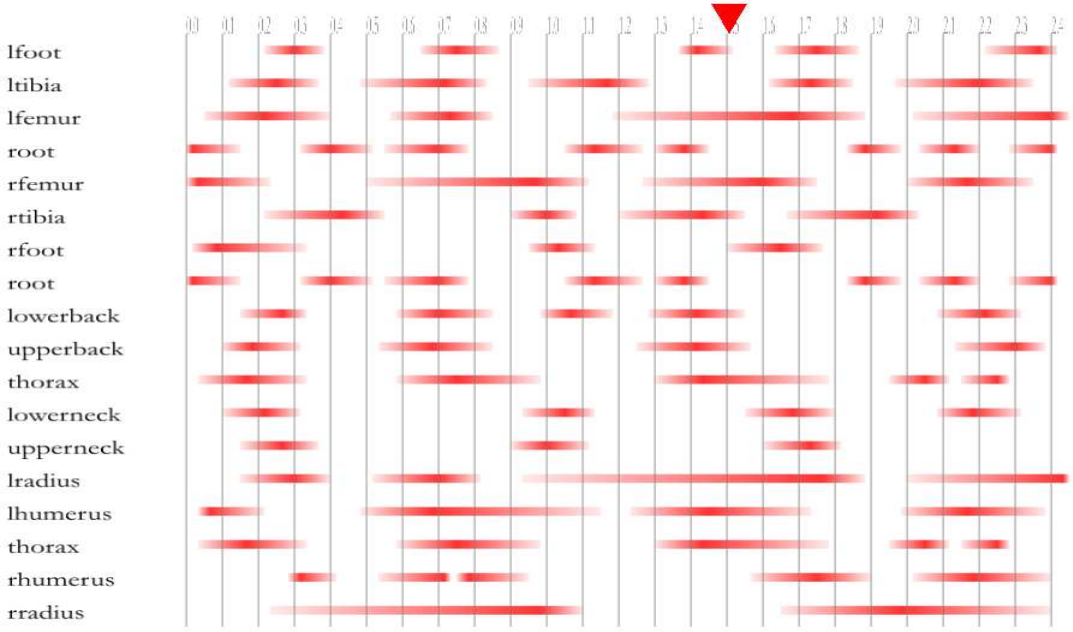

Figure 16: Example of our segmentation result. The red lines represent the segments for each bone.

Figure 16 shows the segments for each bone where the segmentation is performed. The horizontal axis is the time axis. The red lines show where the segments/swings are in time. Therefore, the length of the segments gives us the duration of each swing and their position tells us the order in which they occur in time. What is not shown on is the magnitude of the swing (how wide an angle the swing goes through and the starting and end angle) as well as the axis of rotation of the swing.

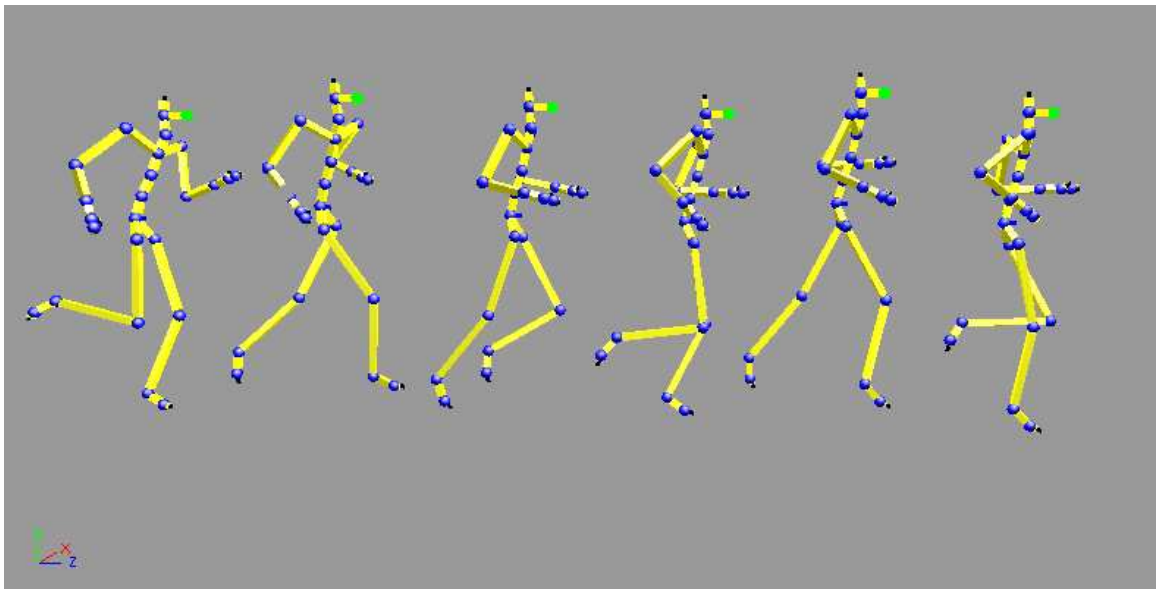

Figure 17: Snapshot of run sequence from playing back the segmented result in Figure 15. Note how the walk motion is preserved.

From the above result, we can see that our segmentation model clearly is able to extract out the segments as defined earlier since playing back the segments yields something close to the original run motion. Recall in the background knowledge in Chapter 2 that animating a skeleton basically means setting the rotations for each bone from a given motion data file as time goes on. With each segment, we have information on the axis of rotation, start and end angle of rotation and the time they occur. Therefore, we can "playback" these segments easily by calculating the rotation for each bone from the segments depending on the current time.

To give a quantitative value to the quality of the segmented motion, we did a frame by frame comparison between each frame from a motion played back from the segments and from the original.

Each frame as we have discussed, is a set of angles describing the rotations for each joint in the skeleton for a particular time. We can think of it as an *n*-dimensional vector where n is the number of angles for each frame. For the case of our motion data which is available one the CMU Graphics Lab site, n is 62 and the angles are represented in degrees instead of radian. So to do a comparison between 2 frames, we just use a simple Euclidean distance between the 2 vectors. The larger the distance, the more different

from each other they are. So for a motion with 100 frames, we find the distance between each corresponding frame and average this distance since it is meaningless to compare absolute distance for a motion due to the different in lengths of motion. We call this the **Segmentation Error**.

From a set of around 50 motions which consists of motions such as running and jumping, we did this comparison between the motion played back using the segments generated and original motion.

The highest Segmentation Error is 40.65˚ while the lowest Segmentation Error is 20.23˚. The average Segmentation Error among these 50 motions is 25.50˚. Therefore, we see that our segmentation produces motions that differ not much from the original motion.

Take note that these numbers mean that on average the sum of all the differences between corresponding angles in a frame from the segmented motion and the original motion is 25.50˚. Since each frame has 62 bones, the average distance between each angle is 25.50 / 62 = 0.411˚.

The **Acceptable Swing Error** we used for generating the results above is 0.4. The value for this can range from 0 to 2, because the **Acceptable Swing Error** is the average error of the calculated swing from a perfectly geodesic swing. The reason for is that the error between each frame from a perfectly geodesic swing is the cosine of the angle between them. If the **Acceptable Swing Error** is set to 0, it means only perfectly geodesic swings will be accepted. On the other hand, if it is set to 2, then the whole motion can be label as a swing.

Since we have no mathematical method to show what the optimal Acceptable Swing Error is, we resort to empirical means by plotting a graph of the average Sum of Differences for the 50 motions against the **Acceptable Swing Error**.

Figure 18: Plot of Average Sum of Differences for the 50 motions against the Acceptable Swing Error used.

We can see that at 0.4, the Average Sum of Differences is the least. Therefore, we chose 0.4 as our threshold value. For value of 0 for the **Acceptable Swing Error**, the high Average Sum of Differences might be due to the fact the segmentation finds no swings and hence the result when compared to the original motion is very different.

4.4 Limitation

However, our segmentation model is not the be all and end all. Even though we could extract the segments quite nicely, there are still missing information to be filled in between each segment which we did not handle. What we do now is to perform interpolation from one segment to the next for the gaps in between each segment. We do this because the spaces in between are usually motions where the joint is almost stationary with not much movement. Therefore, interpolating from the end of one segment to the beginning of the next will give us a good approximation of these “blank” spaces.

Also, our method does not work very well for joint that are slow moving or is quite stationary during a motion. For it to work, the joint must go through a huge movement. As a result, in one motion, some joints might not animate properly through the segmented result.

However, this is only the first step towards building a good segmentation model for motion data but already with this there are a couple of applications which we can use this model for as detailed in Chapter 5.

Chapter 5
Applications

The segmentation result from our model can be used in a number of applications such as motion data compression, motion data indexing and retrieval and motion editing. There may be more applications for this then we can think of now but the segmentation result can definitely be used to further build a hierarchical model for motion data since it represents a higher level representation for the raw motion data below.

5.1 Motion Data Compression

In typical Motion Data compression, the objective is to compress the data to a smaller size as in any other forms of compression.

Using the segmented result from the segmentation model, we already have a good representation of the original motion in a compact form. By just saving the segmented results directly, we can typically achieve a file size of around ¼ of the original motion data. Furthermore, this is a lossy compression; therefore, we cannot expect full recovery of original motion data just from the segmentation results.

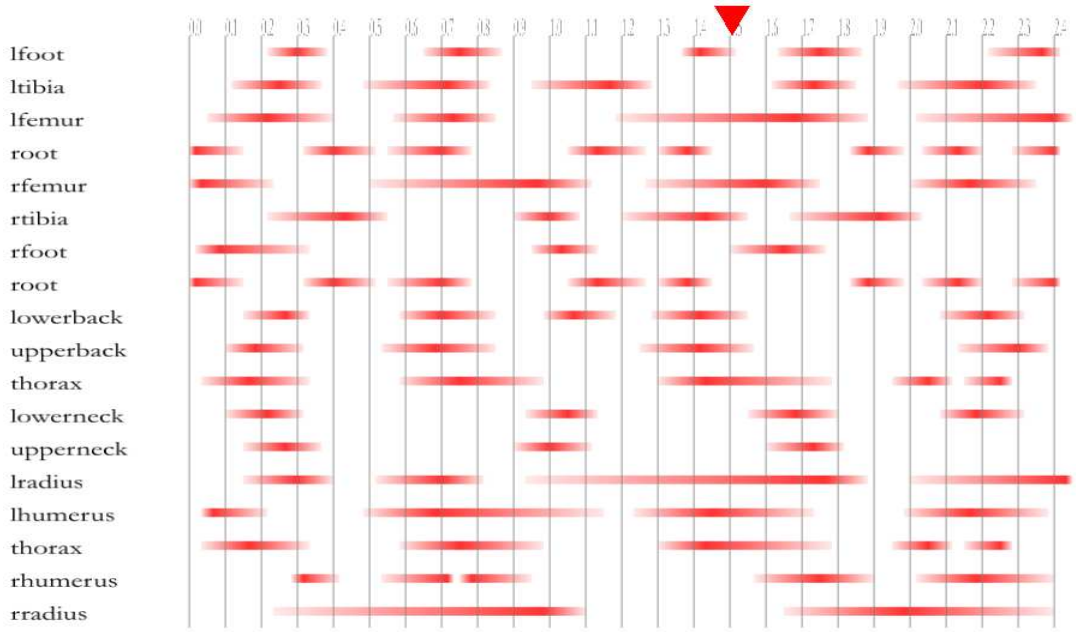

Figure 19: Segmentation result of a running motion.

From the segmentation result in Figure 19, we see that most part of the motion is made up of the horizontal red lines which represent the swings in the motion. Since these swings are replaced by a more compact representation, the amount of data to be store is significantly reduced. Furthermore, the animation quality is not significantly affected.

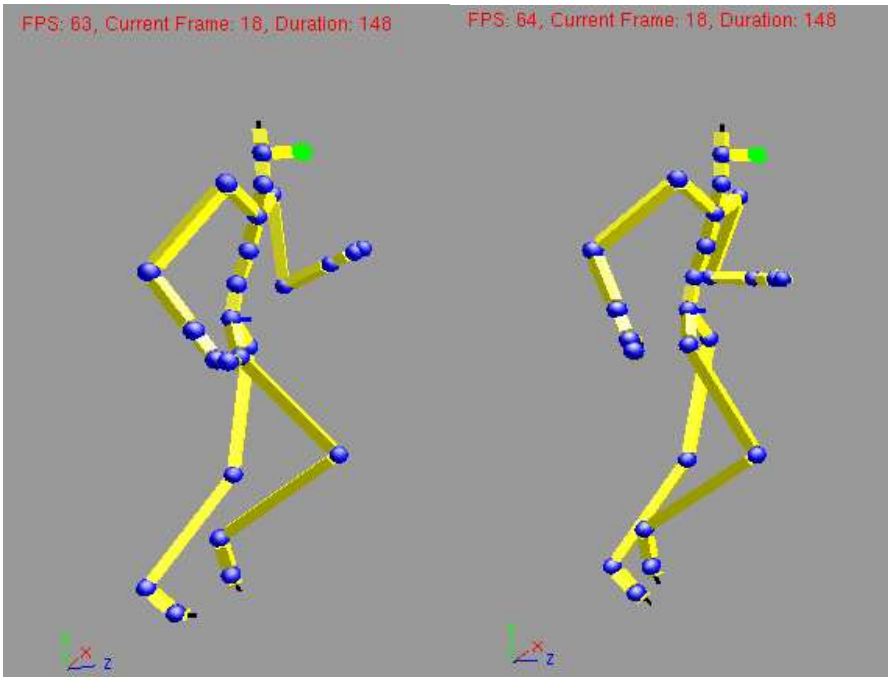

Figure 20: Original motion (left), compressed motion (right) at the same frame for a running motion.

The above image shows the comparison between the original and compressed motion. They only differ slightly in the pose but generally, the original motion is retained in the compressed version.

Here are more comparisons between original and compressed motion.

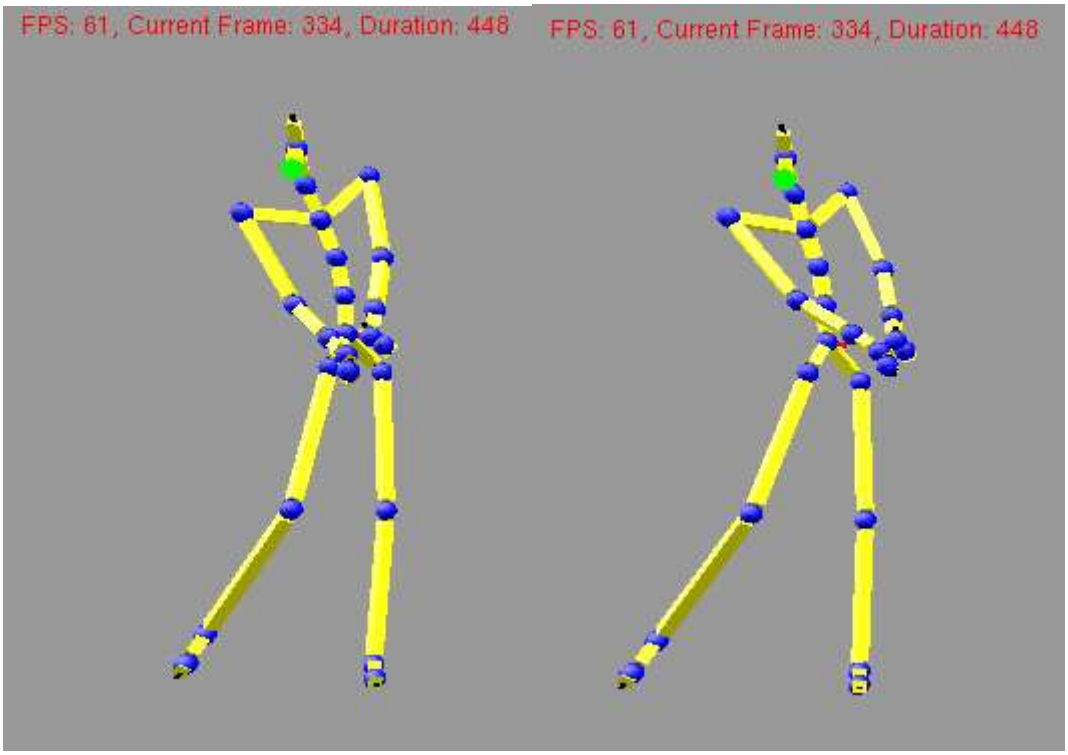

Figure 21: Original motion (left), compressed motion (right) at the same frame for a golf swing motion.

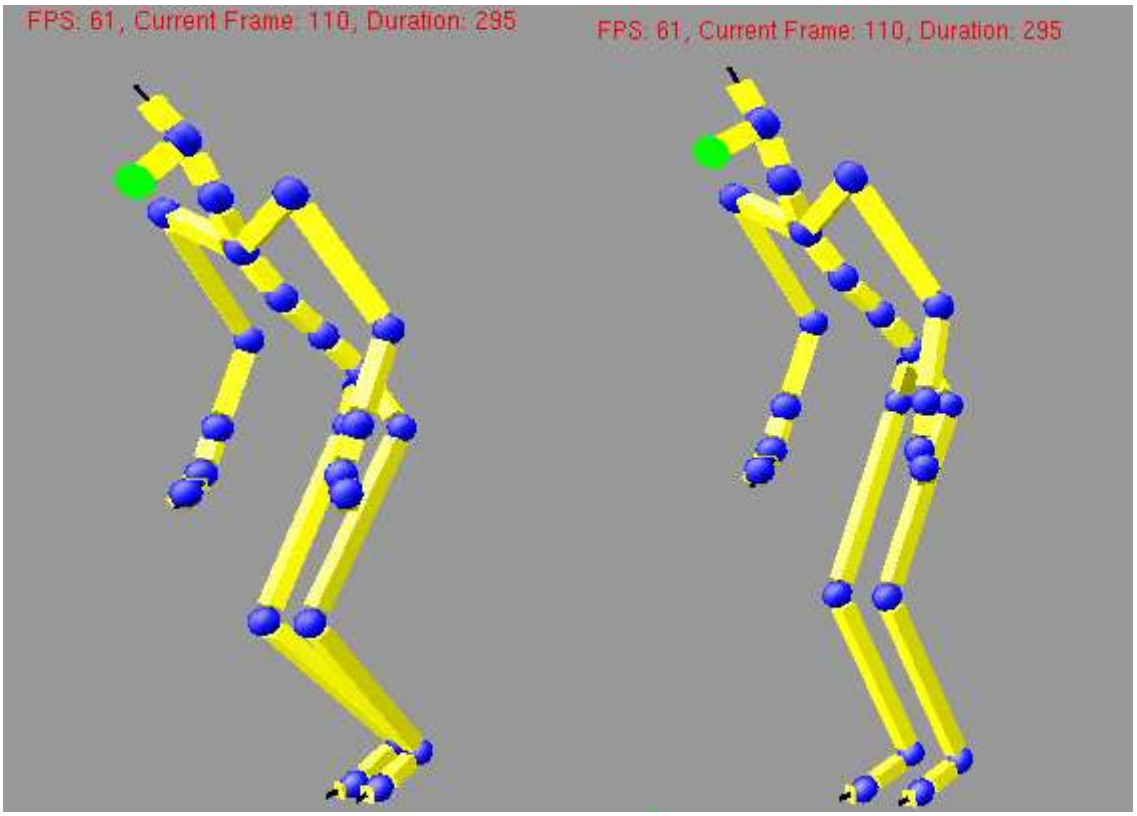

Figure 22: Original motion (left), compressed motion (right) at the same frame for a forward jump motion.

The compressed version could be used in a real time application such as computer games where characters have different level of details (LODs). Normally, the different level of details has different 3D meshes of different number of polygons as well as different level of texture details. We can also have different level of details for animations of such characters as well with our technique. The memory requirement when loaded into memory for the compressed version is smaller than the original motion data and this is crucial for games where memory is often limited.

We tested the compression on around 100 motions in our collection of motion and the compression level is always around ¼ of the original size.

This might not be earth shattering and we consider this to be just a by product of the motion segmentation model. This compression is achieved by just segmenting the motion data. It is simple and easy to implement, compared to other compression methods which might take more time to analyze the motion data before compressing.

In our tests, a typical motion when represented just by the result of segmentation, can be compressed to around ¼ its original size. The compression is a direct result of replacing a sequence of motion data representing a swing with our representation. Note that this is a lossy compression; we can't get back the exact motion data back from the compressed version. Compared to other motion data compression methods [10], our way might not seem much, but note that this compression feature easier to perform.

5.2 Motion Data Indexing and Retrieval

The objective of motion data indexing and retrieval is to be able to some how compare 2 motions with each other to know how similar they are. Since our segmented result gives us the swings of each bone with the time they occur, axis of rotation and other information, this could be used to compare 2 motions. If we have this ability, then give a motion, we can query a database of motion and find motion similar to the given motion.

The segmentation result from our model gives us a rather unique signature for each motion. Moreover, the result is similar for similar motions as shown in the result for 2 running motions below.

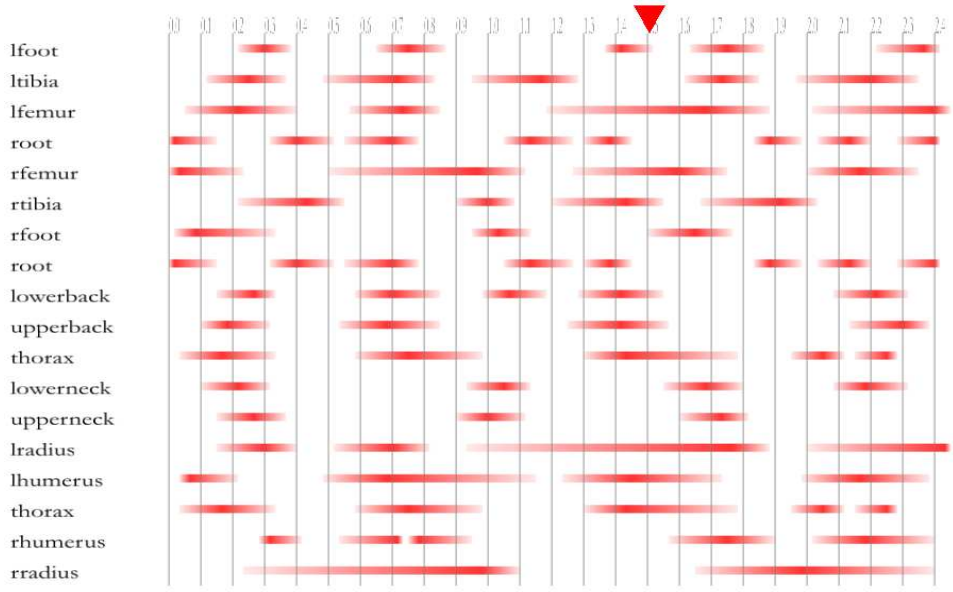

Figure 23: A running motion.

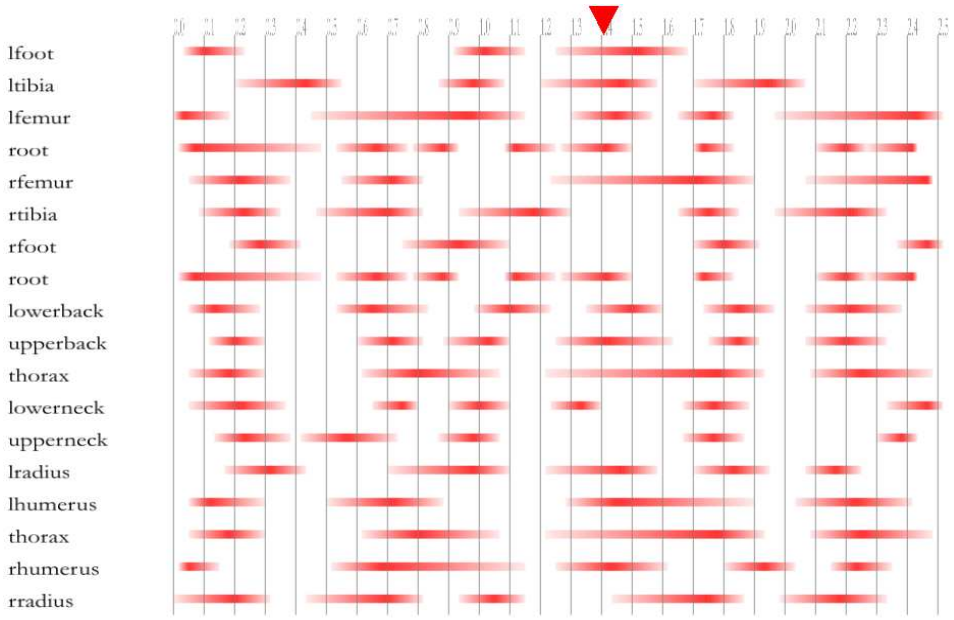

Figure 24: Another running motion.

Therefore, the segmentation result could be used as an index when storing motion data in a database. It can be used for querying the database for similar motions when given a reference motion. We did preliminary work on this by treating the segmentation result as a graph and doing a brute force graph comparison through our collection of motion data and ranking search result based on the similarity. For difference in timing of motion, we overcame that by comparing the segmentation result of the reference motion in a sliding windows fashion against other motions.

In the database of motion that we had, we have 8 types of motions, namely:

- Running
- Jumping
- Golf swings
- Stylized walk/run
- Ball kicking

- Basketball motion
- Baseball throwing
- Punching

For each type we have 20 motions each for a total of 160 motions.

Some results of the querying by using the segmentation result:

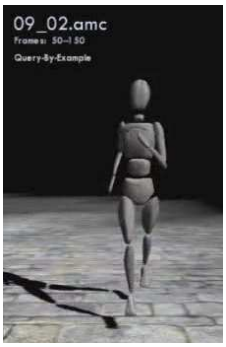

Figure 25: The reference motion.

Figure 26: Top 6 matches from our motion database collection.

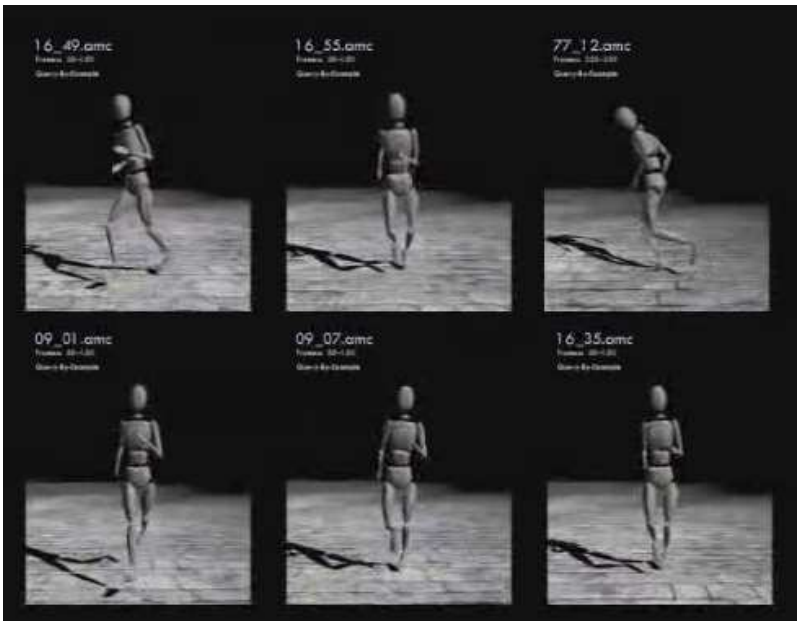

Figure 27: The next 6 matches.

Notice that in the later results, there are walking motions in which the character is slightly leaning to one side, and hence it is less similar to the first 6 motions which are running upright. Therefore, the proof of concept of using the segmentation result for indexing and retrieving motion in a database is workable.

However, because of the brute force nature of the way we compared 2 motions, a better way to simplify the comparison and searching is needed.

One thing we can be sure is that the segmentation result is a very good representation of a motion data clip and it definitely is viable to use it as a base for building a hierarchical model for motion data.

5.3 Motion Editing

For motion editing the main aim is to be able to edit a given motion. As mentioned in the earlier chapters in this thesis, most motion synthesis/editing deals directly with the motion data itself.

However, because our segmentation result represents the underlying motion in terms of the swings, we have a higher level representation of the motion and we can try to modifying these swings to edit a motion instead.

Below is the segmentation result of a motion which consists of a short run before a jump.

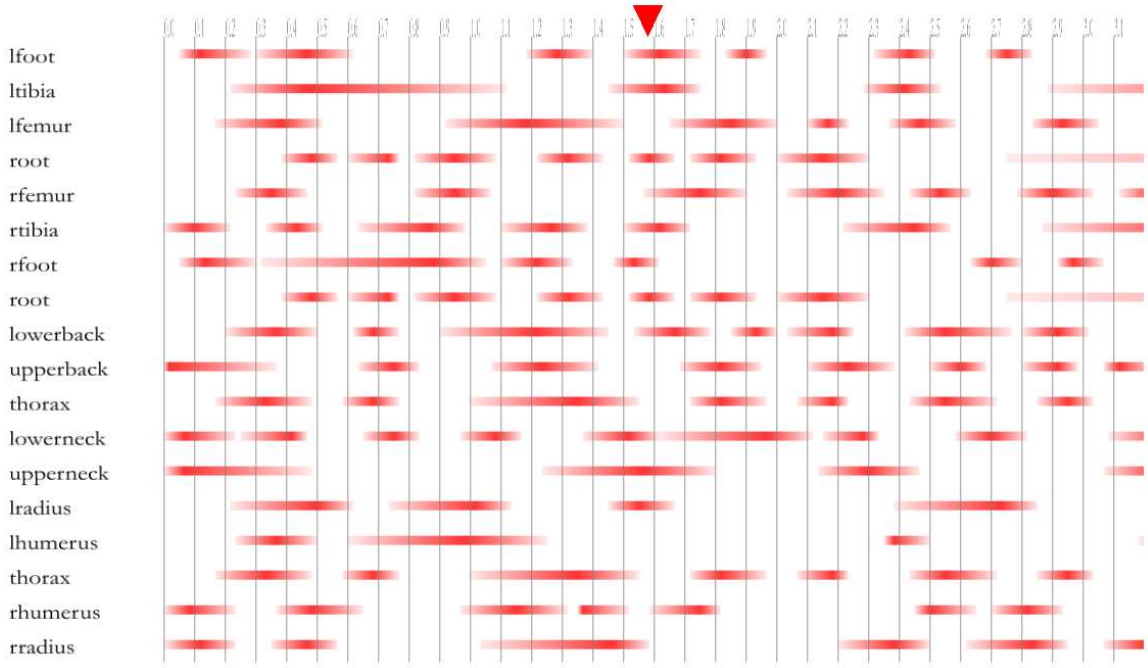

Figure 28: Segments of a short run and jump motion.

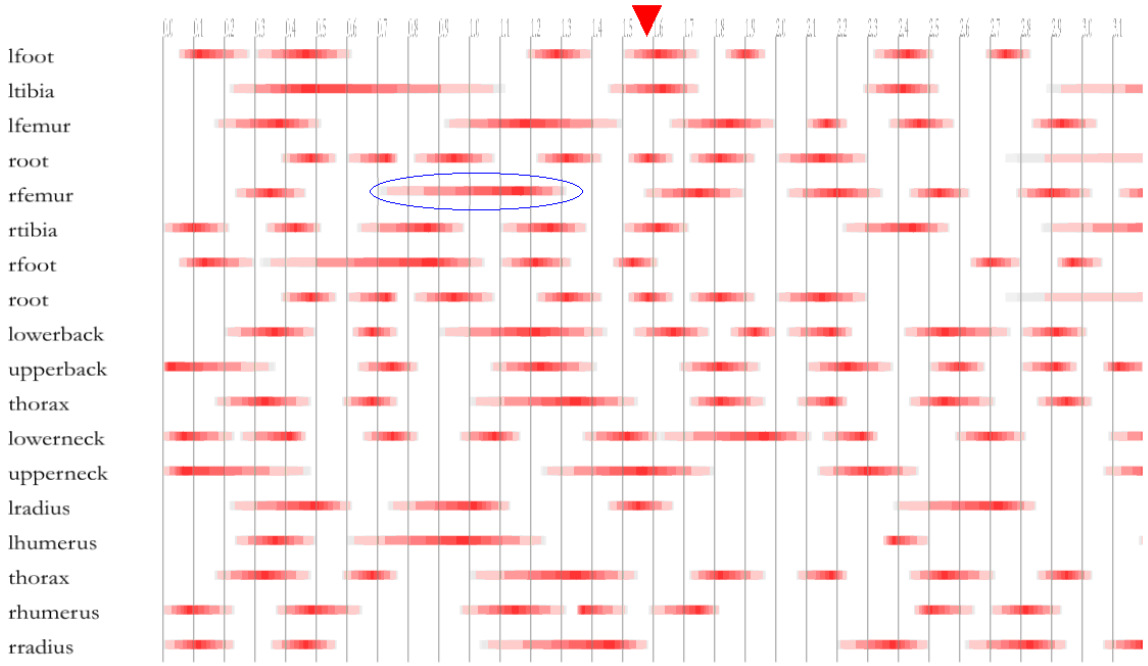

Figure 29: One segment of the rfemur lengthen as highlighted by the blue ellipse.

By lengthening the segments belonging to the legs of the skeletons, the motion becomes more stretched out at the legs.

Below is a comparison between the original and edited motion at the point in time where the edited segment is.

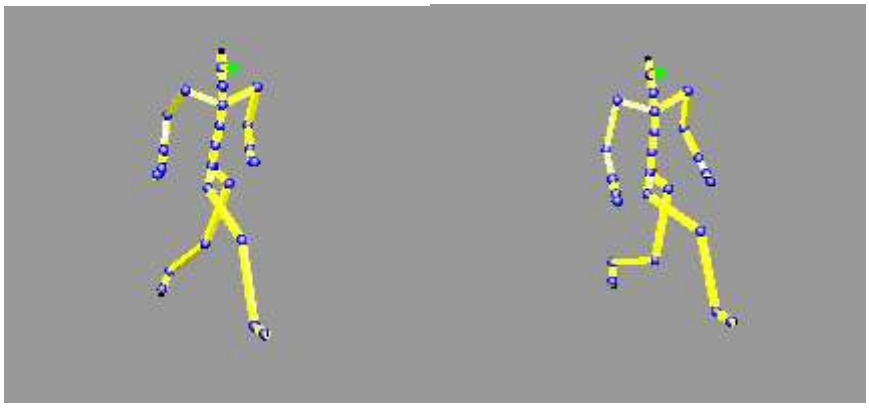

Figure 30: Left (before editing) Right (after editing).

Here are some more examples,

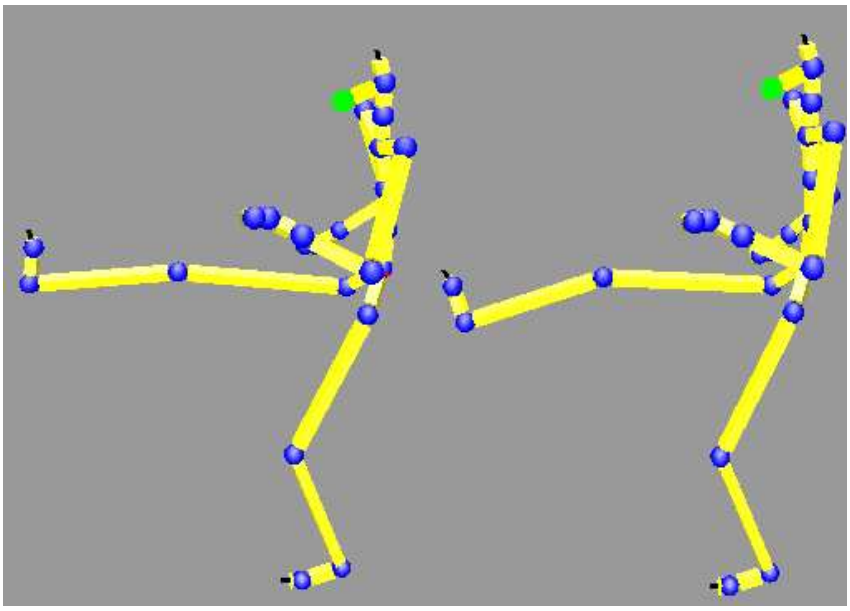

Figure 31: Left (before editing) Right (after editing).

In Figure 31, the one of the segment for the swing of the right leg is edited such that the angle of rotation is decreased, resulting in a slightly shorter kick.

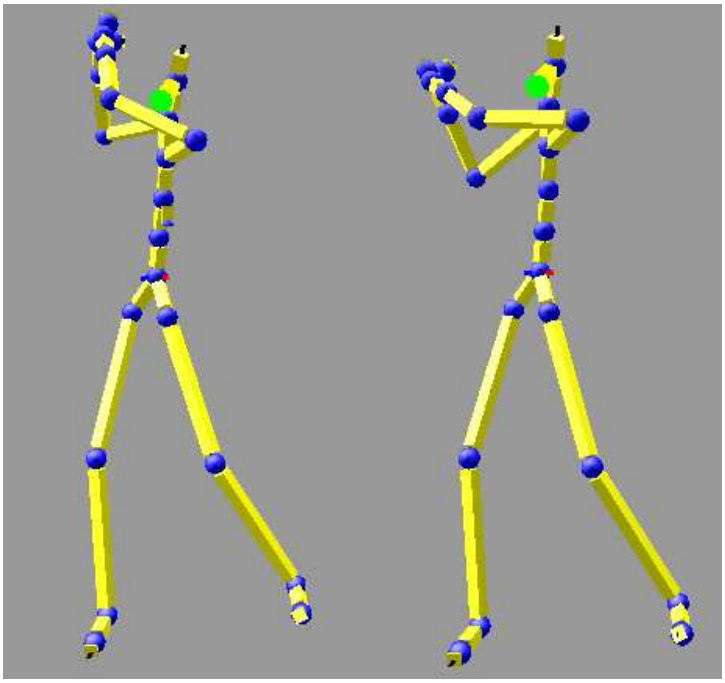

Figure 32: Left (before editing) Right (after editing).

In Figure 32, the segments for the swing of both arms are edited to reduce the angle that they swing through, thus we can see that the arms are now “lower” then the one before editing.

Note that the results might not be good enough for production level editing because our segmentation model does not represent the motion fully. However, this proof of concept shows that if we can refine and improve the model further, a more intuitive way of editing/synthesizing motion might be feasible.

Chapter 6

Conclusion

6.1 Contributions

The main contributions of this project can be summarized as follows:

- We came out with a novel segmentation model with which can represent a given motion at a higher level than the raw motion data based on the swings of the limbs during a motion

- By just playing back the segmentation result and obtaining a rather high quality version of the original motion, it shows that the segmentation model is well formed and we have achieved the main aim of this thesis. We can think of the segmentation result as a lossy compression of the original motion data while retaining the main characteristics of the original motion.

- Motion Indexing and Retrieval can be done using our segmentation model for motion data. Our result shows that a search given a query motion into a collection of motion yields pretty good results.

- From the segmentation result, simple motion editing can be done at a more intuitive level then simply manipulating the raw motion data itself. Although the method could be refined by combining with physical properties such as the mass of joints and hence the center of mass to calculate the whether the balance of the character when the motion is edited.

References

[1] KOVAR, L., GLEICHER, M., PIGHIN F. 2002. Motion Graphs. *In SIGGRAPH 2002.*

[2] JING W., BODOHEIMER B. 2003. An Evaluation of a Cost Metric for Selecting Transitions between Motion Segments. *In Eurographics/SIGGRAPH Symposium on Computer Animation 2002.*

[3] ARIKAN, O., FORSYTHE, D. 2002. Interactive Motion Generation from Examples. *In Proceedings of SIGGRAPH 2002.*

[4] KOVAR, L., GLEICHER, M. 2003. Flexible Automatic Motion Blending with Registration Curves. *In Eurographics/SIGGRAPH Symposium on Computer Animation 2003*

[5] KOVAR, L., GLEICHER, M. 2004. Automated Extraction and Parameterization of Motions in Large Data Sets. *In SIGGRAPH 2004.*

[6] ARIKAN, O., FORSYTHE, D., O'Brien, J. 2003. Motion Synthesis from Annotation. *In Proceedings of SIGGRAPH 2003.*

[7] FRAZEE, P. Skeletal Animation. Article on http://www.gamedev.net. http://nehe.gamedev.net/data/articles/article.asp?article=03

[8] NEHE PRODUCTIONS! Lesson 31. Tutorial Articles on attaching a mesh to a skeleton. Found on http://www.gamedev.net. (http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=31)

[9] ARIKAN, O, Compression of Motion Capture Databases. *In Proceedings of SIGGRAPH 2006*

[10] SIDDHARTHA, C, SUCHENDRA, M, KANG, L, Human Motion Capture Data Compression by Model-Based Indexing. *IEEE Transactions on Visualization and Computer Graphics 2007*

[11] WANG, J, DRUCKER, S.M, AGRAWALA, M, and COHEN, F.M. The Cartoon Animation Filter. *In Proceedings of SIGGRAPH 2006.*

[12] KWON, J, and, LEE, I, Rubber-like Exaggeration for Character Animation, *Pacific Graphics 2007*

[13] HSU, E, SILVA, M, and, POPOVIC, J, Guided Time Warping for Motion Edition, *Symposium of Computer Animation 2007*

[14] LASSETER, J, Principles of Traditional Animation Applied to 3D Computer Animation, *SIGGRAPH 1987*

[15] BRUDERLIN, A, and, WILLIAMS, L, Motion Signal Processing, *SIGGRAPH 1995*

[16] INAM, R, IDDO, D, VICTORIA, C, DAVID, D, and, PETER, S, Multiscale Representations for Manifold-Valued Data, *Multiscale Modeling and Simulation Volume 4, Issue 4, pp. 1201-1232, 2005*

[17] GRASSIA, F. SEBASTIAN, Practical Parameterization of Rotations Using the Exponential Map, *Journal of Graphics Tools, volume 3.3, 1998.*

[18] Euler Angles Online Article at Wikipedia. http://en.wikipedia.org/wiki/Euler_angles

[19] Rotation Matrix Article at Wikipedia. http://en.wikipedia.org/wiki/Rotation_matrix

[20] Quaternion Articles at Wikipedia. http://en.wikipedia.org/wiki/Quaternion

[21] Gimbal Lock Explanation at Wikipedia. http://en.wikipedia.org/wiki/Gimbal_lock