Donald J. Macleod

MSc Computer Animation 2006

Masters Project

Skeletal Character Animation within a Virtual Environment

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Background

With the significant advances in the computer graphics and animation field over the years, greater demands have been placed on the quality and realism of character animation within today's interactive graphical environments.

Two main goals for providing better quality character animation are to provide realistic motion of the character's inherent structure, and to apply realistic deformation of the character's surface in response to these motions.

For interactive applications these two goals also must be balanced against processing and memory limitations so that the application can run at a sufficiently high frame-rate for the user to interact with when it is carrying out the character animation.

The complexity of capturing sets of realistic character motion has significantly reduced with improvements and advances being made in character simulation within commercial animation software packages and technologies such as motion capture.

However the nature of many virtual environments (such as environments within today's video-games, or training simulations) can require characters to assume a possibly infinite number of possible poses in its animation within the environment (for example a walking motion may need to be adapted to rough terrain requiring the walk motion to be modified to ensure the feet do not move through the ground). Attempting to capture all the animation poses for every possible scenario within such rich virtual environments would be an infeasible task.

In order to tackle such issues a great deal of research has been carried out on real-time motion editing techniques to apply pre-existing motion data to individual situations requiring specific needs, with the goal of having the edited motion maintain the integrity of the original motion as much as possible. Similarly, a number of methods and techniques have been developed to apply realistic deformation of characters according to their motion. The following two sections in this chapter give a general overview of some of the key aspects of on-line character motion editing and character deformation.

1.1.1 Motion Editing

We shall give a general overview of the following important areas of motion editing in this section - motion blending/interpolation, applying motion data to new environments and characters, and the application of constraints. The specific techniques discussed in this section that have actually been used in this project are also discussed in more detail in Chapter 2.

In order to create streams of motion within the virtual environment current applications generally assemble clips of motion which have been sourced from either motion capture data or key-framed animation sequences generated in commercial software packages (such as SoftImage|XSI, Maya, Houdini, Endorphin etc.).

The assembly process generally requires generating a directed graph structure to represent the possible flows of sequential motion within the virtual environment where the graph edges represent particular segments of motion and the graph nodes represent the choice points connecting the motion segments.

A common approach in computer games is to create such a directed graph manually. These graphs are typically referred to as move trees [4]. Commonly, rather than procedurally transitioning between the motions in the move tree, an animator will author transition animation sequences for the transitions between motions.

Using motion editing techniques however transition sequences can be generated procedurally for smoothly transitioning between motions that can validly be transitioned between within the virtual environment. The difficulty of transitioning between two motions depends on the similarities between the two motions. In practice if the two motions are quite similar we can use simple blending techniques such as linear blends on the joint orientations between relative stages of the motions to generate these transitions. For particularly difficult transitions (or if a transition is to be carried out in a particular style) sometimes it may be required for an animator to hand-craft a transition motion sequence or have a transition sequence motion captured (which itself will require a motion editing process to be applied to it).

There are a number of existing techniques that allow for the motion graph structure to be generated automatically from just the motion data. Techniques such as Motion Graphs [1] and Snap-Together Motion (STM) [3] automatically create graphs from the original motion capture data by carrying out comparisons between frames within the motion data to automatically detect transition points (with some user input such as a threshold on the difference between matched transition frames).

In comparing two frames for whether they match or not the system generates a character pose point cloud for a neighbourhood of frames centred at the particular frame being checked and calculates a 2D translation along the floor plane and a rotation in the vertical axis to be applied to the second frame neighbourhood so that the two frame neighbourhoods are as closely matching as possible. A neighbourhood of frames is used rather than just the single frames to incorporate derivative information into the match comparison. As a result similar motions captured in different positions and orientations can be paired as transition points and the motions can be sequenced together. Figure 1 below illustrates two running motions in different directions being matched using the algorithm in these techniques.

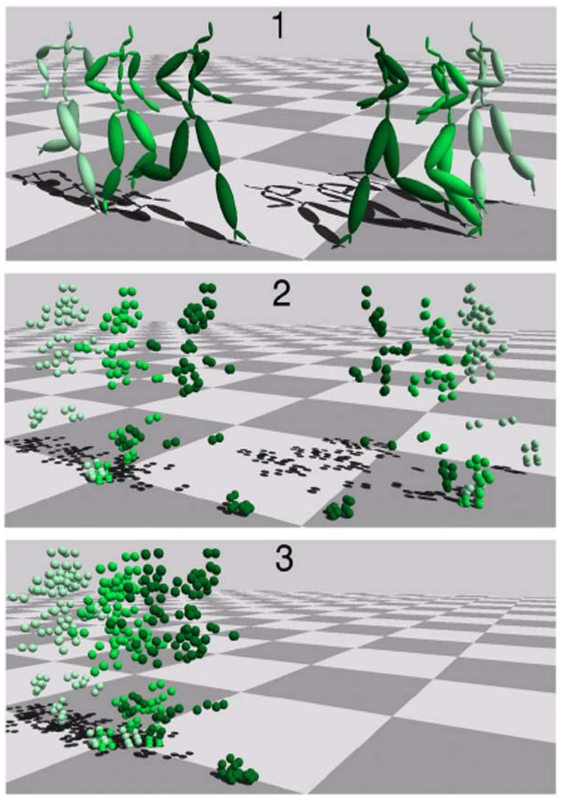

Figure 1: Comparing frames from similar motions in different directions by applying a translation along the floor plane and a rotation around the vertical axis and subsequently carrying out an optimal sum of least squared distances between corresponding points [3].

Before discussing the creation of new animation sequences (including transitional sequences) we shall distinguish between motion blending and motion interpolation.

We use the term motion blending here for skeletal animation to specify any motion editing technique whereby frames for new motions are generated from multiple existing motions using time-varying weights for each motion contribution. Motion interpolation similarly involves the blending of multiple motions to create new motions but with the weights being specified explicitly (as opposed to being defined by a function of time) to create “in-between” motion sequences.

Motion blending is useful for generating smooth transition sequences and to apply adjustments to motions in a smooth continuous fashion. Motion interpolation techniques are useful for combining two or more existing motions together to create individual new motions. For example a walk motion on flat terrain could be combined with a severe limping motion to produce a walk animation with a slight limp, or an arm's motion from a waving sequence could be transplanted onto a running motion sequence to create a waving run motion [5].

Note that the motion weight values can be assigned individually at the joint level but generally the weights are assigned at the skeleton level so that joint transform contributions from a particular motion frame are the same throughout the skeleton. Assigning weights at the joint level can be useful when we wish to take separate motions of individual segments of the skeleton (such as an arm) from some of the motions in the blend. Unless otherwise stated when we refer to blend weights in this paper we shall be referring to weights assigned at the skeleton level.

The quality of new motion sequences generated from blending/interpolation are largely dependant on how much information about the motions is supplied to the blending algorithm. If no information other than the raw motion parameters (translations and orientations of joints) is supplied then, unless the motions being blended are closely matched and in sequence with each other, poor results will be generated from simply blending the frames from the same time period within the motion sequences.

Generally for blending and interpolation between motions, information on how the motions correspond to each other in timing terms is required so that we can retrieve transform data from roughly equivalent times in the motions (e.g. leg cross-over times for walk, run and sneak motions)

This information may be specified manually by user input or calculated by an automatic process (as with Kovar et al's Registration Curves [6], for example).

Generating streams of motion in an interactive environment will generally require motion to be edited based also on additional environmental data rather than solely generating motion using the raw animation data. For example we may want to apply a punch motion that has been captured but need to redirect the punch to a specific location within the environment (e.g. the face of an opponent) or we may in fact be applying the same motion data to a number of characters within the environment which have differences in their skeletal structure (and hence different geometric reactions to joint rotations).

Solving such issues requires an Inverse Kinematics (IK) solver in one form or another to solve the geometric constraints on a character pose at the specific time instances (e.g. ensuring feet land on ground properly for each step).

The methods for IK solvers in graphics fall into two categories; geometric or analytic, or numeric or iterative solutions.

Analytic methods use closed-form geometric constructions to compute configurations for end-effector position directly. When analytic methods are used on an under-constrained system the methods must be carefully constructed for the particular type of joint structure that the solver is used for. (Note by under-constrained here we are referring to an IK system where the DOF of the joints are greater than the DOF of the end-effector solutions. Usually for a single end-effector solution there are 3 or 6 DOF, 3 for world-space translation and a possible additional 3 for the orientation).

Analytic solvers for 7 DOF [7] and 12 DOF [8] joint chains have been developed which can be incorporated to separately solve for individual limbs of a skeleton.

Numeric solvers use equation solving or optimisation techniques to solve the IK problem and provide a more general IK solution for skeletal structures. In contrast to analytic solvers, numeric solvers can be applied to under-constrained skeleton structures with many DOF. Because of the general non-linear nature of the equations, numerical solvers generally require the solving process to be carried out iteratively until the end-effector is sufficiently close to the target (or as an exit strategy, exit when either a maximum number of iterations have been executed or the end-effector adjustments reach a certain lower error limit toward the desired solution).

The advantages of analytic IK methods are that they find solutions more quickly than numerical solutions and they are guaranteed to find a solution if it exists. They do however lack flexibility in how they choose solutions and have the drawback that a solver of this type can generally only be applied to a particular type of joint structure (such as having a 7 DOF solver for arms and legs).

Some IK solvers use a hybrid of analytic and numeric methods, using analytic methods to solve separate joint chains such as arms and legs where a closed form solution is available and using numeric methods to solve the computation of body posture (such as with [9] and [10]).

Numerous IK solving techniques have been developed such as Cyclic Coordinate Descent (CCD) (and a damped CCD variant), pseudo-inverse, Jacobian transpose, damped-least-squares (DLS), selectively damped-least-squares (SDLS) among others. For details of these techniques the reader is referred to [11], [12], [13], and [14].

In addition to the spatial constraints motion editing techniques must also take temporal properties into consideration. For example the high frequency snap of a punch motion should not be dulled on editing to redirect a punch. There are numerous different approaches taken by constraint-based motion editing methods for handling the temporal constraints of motion. We refer the reader [15], which discusses some of the available methods such as Motion Warping, Per-Frame Inverse Kinematics (PFIK), Per-Frame Inverse Kinematics plus Filtering (PFIK+F), and Spacetime Constraints.

Figures 2 and 3 below shows results from work done by Micheal Gleicher [16] that solves the spatial and temporal constraints of a motion, using a Spacetime Constraints solver, to adapt the motion to characters of different size and skeletal shape preserving the frequency characteristics and special features of the original motion.

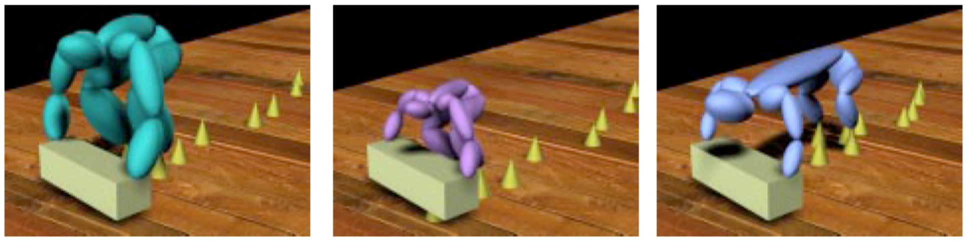

Figure 2: Results from using Spacetime Constraints motion editing method on a pick-up motion for different sized characters. Left shows the motion frame on the original actress, centre shows the motion frame for a character 60% of the size of the original, and right shows the motion frame for a character with very short legs and arms and a long body sized characters picking up an object [16].

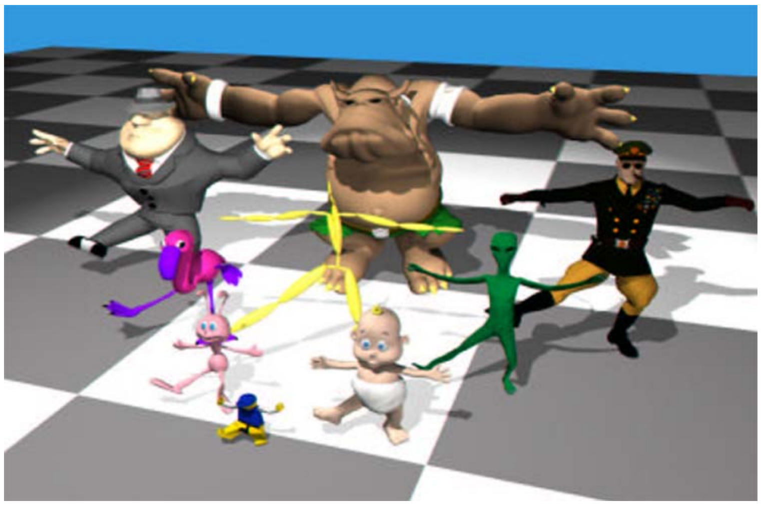

Figure 3: Results from using Spacetime Constraints retargeting a dancing motion to a variety of Characters [16].

1.1.2 Character Deformation

In addition to applying realistic motion to the character models an important aspect of the character animation is the deformation of the character's body in relation to its motion. Generally it is not plausible for an animator to sculpt entire meshes by hand for every possible pose of a character. Techniques such as skeletal-based animation are used to aid the process of creating large numbers of mesh variations for animation purposes. For certain cases such as facial animation non-skeletal-based techniques such as blend shapes (also known as Shape Interpolation) are also commonly used but they are not as well suited to the deformation of entire character meshes as the skeletal-based techniques.

Within skeletal-based deformation systems an underlying hierarchical skeleton is attached to the mesh of the character so that as the skeleton is manipulated the mesh of the character is automatically modified accordingly. This attachment of the character mesh to an underlying skeletal hierarchy is called a skin and can be viewed as a function that maps the skeleton parameters (or degrees of freedom) to a deformation field of the mesh vertices.

There are two fundamental aspects of skin creation – authoring and computation. Skin authoring refers to the process of an animator using tool sets to describe the deformation of the skin as the underlying skeleton moves. Skin computation refers to the method by which the character skin is evaluated in terms of the position of the underlying skeleton. For off-line creation of high-end character animation authoring methods are used for the skin creation process, whilst for interactive applications computational methods are used. We will therefore concentrate only on the computational methods that are available.

Due to its relatively fast computation speed and low memory requirements the most commonly used skeletal-based deformation technique within interactive applications such as videogames is a technique known as vertex blending (other commonly used names for the technique are linear blend skinning, Skeletal Subspace Deformation (SSD), enveloping, and smooth skinning).

The process involves assigning a list of joint weights to each vertex, each joint weight representing how much influence a specified joint has on the position of the vertex. The weights are typically assigned according to the vertex distance from the joint in the bind pose position. The number of weights per vertex is usually limited to an upper bound value (usually 1 to 4) and an upper bound may be set on the distance between vertex and joint where a weight may be assigned. The weights for each vertex are then normalised to sum up to 1 and the vertex position and normal are updated according to the current pose joint transforms relative to the joint transforms in the bind pose position.

We denote **v** as the vertex position in the bind pose position, **v**' the position of the corresponding vertex for the adjusted current character pose, Mi the homogenous 4x4 matrix local to world transformation for joint i in the current pose, Bi and wi the weight assigned to the vertex for joint i. Then we calculate **v**' as:

$$\mathbf{v}' = \left(\sum_i w_i \, M_i \, B_i^{-1} \right)\mathbf{v} \quad \text{(Equation 1)}$$

We can similarly calculate the normals of the current pose by taking the inverse transform of the above summed matrix with the translation elements removed, and applying it to the corresponding normal in the bind pose position.

Figure 1 below shows how the process affects the positioning of the vertices on joint rotation. The example on the left shows the vertex positions before rotation. In the centre example the blue vertices are fully associated with the parent joint and the green and white vertices fully associated with the child joint. The result of the rotation creases badly on the inside of the elbow even with this relatively small amount of rotation. The vertex blending example is shown on the right where the vertices have a weighted association with each of the joints. The white vertex in this example has an equal weighting of 0.5 with each joint, the red circles show the positions found by transforming the original vertex position by each of the joint transforms. The final vertex position is found by carrying out a weighted average of the red positions (the weighting of the average in this case being 50-50). The result using the vertex blending method is much smoother and aesthetically pleasing than the example in the centre.

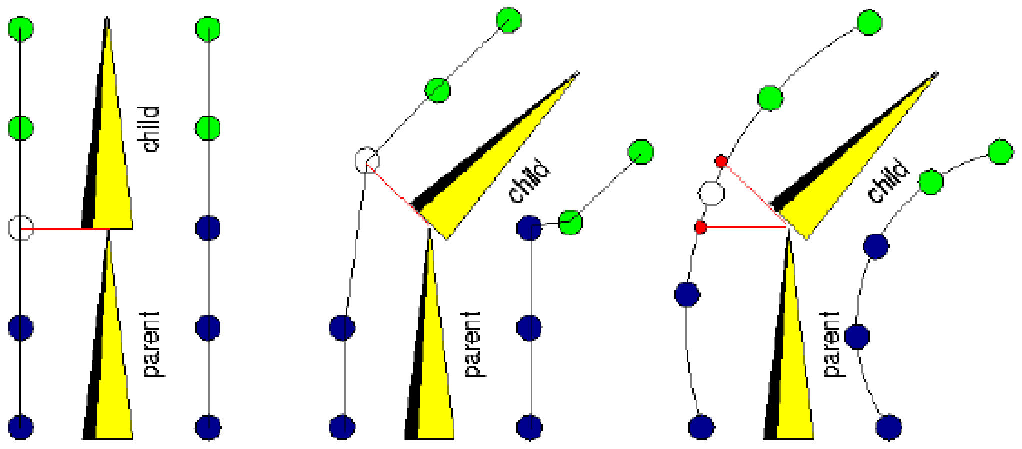

Figure 4: Vertex Blending: (Left) Vertices before rotation. (Centre) Result after rotation without vertex blending. (Right) Result after rotation with vertex blending.

The vertex blending method is a relatively very fast method and it does produce good results for reasonably small rotations. However larger angles can cause serious artifacts. One well known artefact that can occur with this method is the "candy-wrapper" effect when a twist near 180 degrees is applied to a joint. Figure 2 below illustrates this example. On applying the child rotation the vertices are on the opposite side of the bone and so the vertices with a 50-50 weighting are averaged equally between the two opposing positions resulting in the vertex being positioned toward the joint.

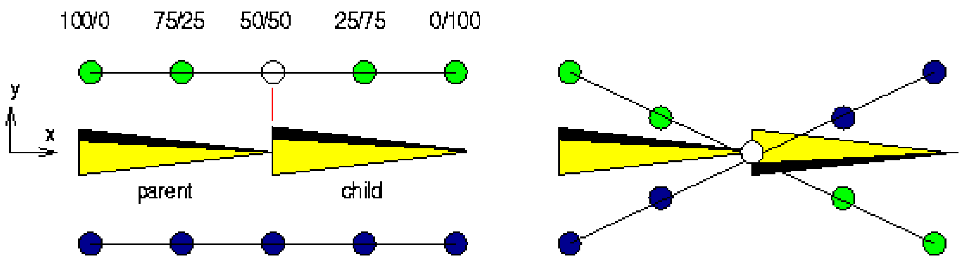

Figure 5: Effect of twisting child joint 180 degrees using vertex blending method resulting in "candy-wrapper" artifact.

Similar displeasing artifacts occur when rotating near 180 degrees around the other two axes. Figure 6 below shows such artifacts on the skin of an arm.

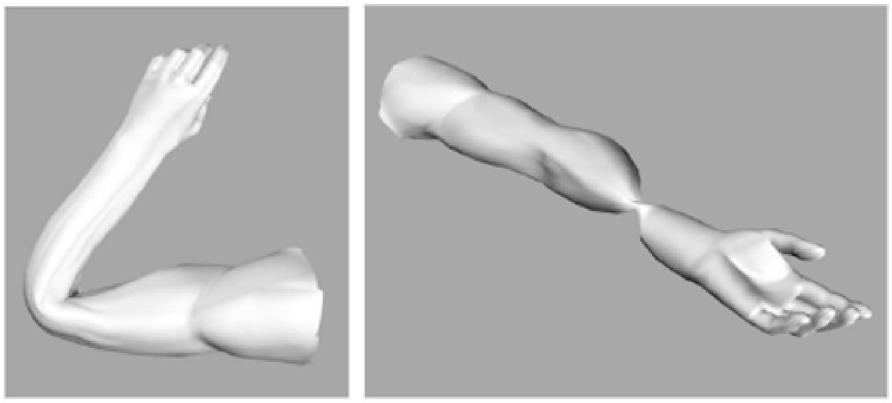

Figure 6: Common artifacts using vertex blending. Left image shows shrinkage around bent elbow. Right image shows twisting rotation on elbow causing collapse of elbow.

When joints may go through such extreme rotations artifacts with the vertex blending method can be alleviated by adding additional joints near the problem joints. In some cases problems may be fixed just by careful re-assignment of vertex weights (for example chest vertices may bulge out on raising arms, lowering the weights assigned to the shoulder joint can alleviate this problem).

Some techniques rely on using more than one sculptured reference pose for their deformation calculations. More and Gleicher [17] extend the vertex blending technique to take in a number of example sculpted poses paired with a corresponding skeleton pose. Their system uses the multiple poses to calculate the weights and vertex positions for the bind pose by finding the values that minimize the least-squares difference between the skin and the examples at all the example skeletal configurations. The system also allows for automatic creation of joints to help solve artifacts such as collapsing with linear vertex blending. Figure 7 below results from using their system on an arm model.

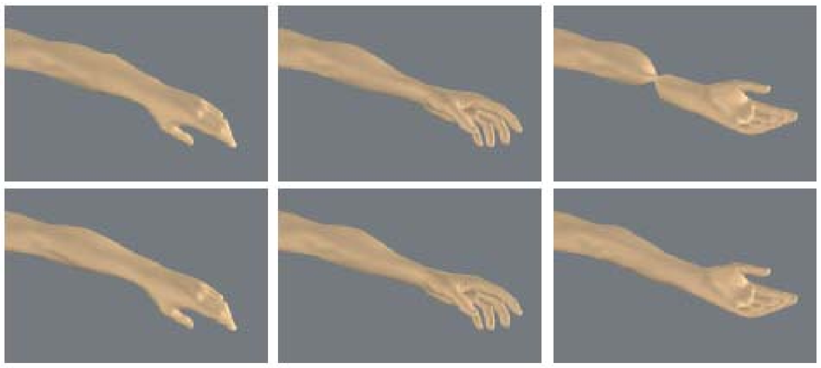

Figure 7: Top row shows typical candy-wrapper artifact using normal vertex blending technique. Bottom row shows the results of using the system proposed by More and Gleicher [17] on the same model.

Other notable techniques include Pose Space Deformation (PSD) and Eigenskin which shall not be discussed here but the reader is referred to [18] and [19] for details.

CHAPTER 2: PROJECT OVERVIEW

2.1 Project Aims

The project aims to develop an interactive application that allows the user to control a virtual character with seamless transitioning between a set of motions.

It was decided at an early stage that the application would make use of motion captured data and use this motion data to provide continuous streams of motion that the user can interact with.

The specific aims of the project have gone through some modifications since the initial proposal. The initial proposal was to incorporate linear blending of the motions with a pseudo-inverse Jacobian IK solver for applying particular constraints to the motion such as adjusting the hit point of a punch or kick.

After more research was carried out in the area of motion editing it was decided however to attempt to incorporate the techniques described in the Registration Curves work carried out by Kovar and Gleicher[6]. Depending on progress of implementing this another goal that was set was to incorporate the 7-DOF analytic IK solver used by Kovar, Schreiner and Gleicher to maintain foot constraints and avoid footskate on blended and interpolated motions[2].

2.2 Tools and Libraries

The application was developed in C++ using OpenGL and the freeglut library for the rendering and user interaction processing. The incorporation of a graphical user interface was not set out as a specific goal for the project so the freeglut library was chosen as it was sufficient for the simple interface incorporated.

After discussions with the staff at the Bournemouth University AccessMocap motion capture studio, regarding file formats for the motion capture data, it was decided to import the FBX motion data files into Maya and use the existing Maya API exporter that was part of the Major Animation project carried out earlier in the year. The Maya API exported text files were parsed with existing C++ code written for this previous project.

The geometry of the character was donated by an MA Computer Animation student Xian Li who provided an .obj file containing the vertex, normal, and texture data for the t-pose of a relatively low resolution character mesh. This mesh was rigged and skinned in Maya, and it was necessary for this project to add extra locked joints to the skeleton to reduce artifacts on some of the extreme motions such as high kicks.

In addition to the text parsing to import the Maya API exported character data it was clear that a number of additional text files would be required to be imported for this project and it was decided that rather than writing numerous C++ parsers for each specific task, an XML parser would be used for the data import. An open-source light-weighted C++ XML parser written by Frank Vanden Berghen was chosen for this task [20].

CHAPTER 3: SOLUTION

This chapter shall discuss the main algorithms used for matching the set of motions, calculating the alignment and blending of motion generated from multiple animations, and transitioning from blended/interpolated motion to non-matched dissimilar motions (e.g. jog to high kick).

3.1 Constructing Dynamic Time-Warp Curve

The first step in the construction of the time-warp curve, where each point on the curve specifies matching times for each motion, is to use a suitable distance metric for calculating the similarity between any two frames of motion on different animations.

It is important for the distance metric not only to incorporate body posture similarities between frames but also take into account joint velocities and accelerations and higher order derivatives.

The distance metric should also take into account that equal joint orientation changes may have significantly different affects on the pose of the model depending on the orientation of other joints in the skeleton. For example Figure 8 below shows the different affect that the same small shoulder rotation has on the position of the arm of a model where the elbow is oriented differently.

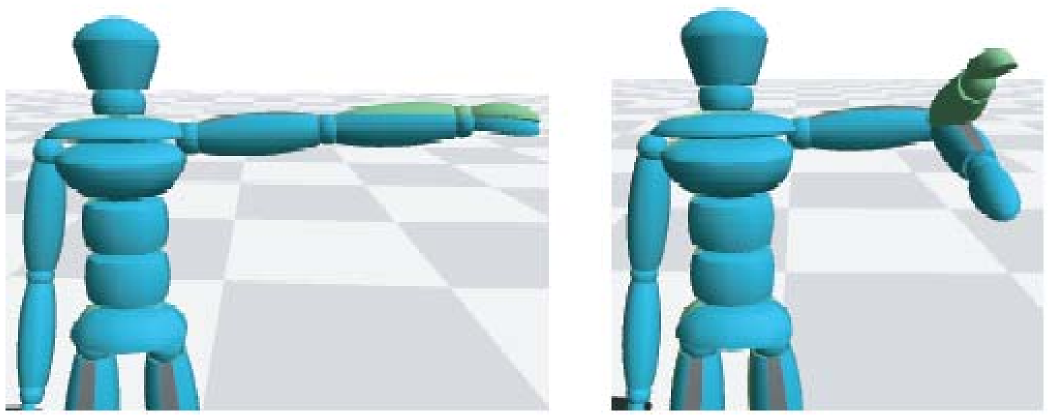

Figure 8: Twisting the shoulder by 30 degrees has a significantly different impact on the posture of the model when the arm is straight (left) and the arm is bent (right).

Another important factor for the distance metric is that motions are fundamentally unchanged by translation along the floor plane and rotation in the vertical axis, the distance metric should therefore be invariant under such 2D transformations of motions.

The distance metric involving frame neighbourhood point clouds mentioned in section 1.1.1 used by Gleicher et al [1], [3], [6] satisfies these requirements and this was the metric used in this project's application. Figure 1 in section 1.1.1 illustrates point clouds being used in the matching of 2 run motions that are similar under a 2D translation along the floor plane and rotation in the vertical axis.

The simpler case of generating point clouds with points located at joint positions rather than located on parts of the skin was chosen in order to save calculation time. Each joint was given a particular weighting (some of which may be zero but none of which were negative), with this information being specified in an XML file that the application imported.

If there are nm marked joints for each pose then each neighbourhood of size 2L + 1 has $n_p = n_m(2L + 1)$ marker points each. The distance between two point clouds is defined as the sum of the squared Euclidean distances between corresponding points on the two point cloud, minimized by all translations along the floor plane and rotations in the vertical axis:

$$D(\mathbf{M_i}, \mathbf{M'_j}) = \min_{\theta, x_0, z_0} \sum_{k=1}^{n_p} w_k \|\mathbf{p_k} - \mathbf{T}_{\theta,\mathbf{x_0},\mathbf{z_0}} \mathbf{p'_k}\|^2$$

(Equation 2)

where p_k and p_k' denote the k'th point in the clouds for the motions Mi and M_j respectively and T represents a rigid 2D transform of a rotation theta in the vertical axis followed by a translation of (x0, z0) along the floor plane. The w_k values are used to preferentially weight certain markers.

A closed form solution for the optimal 2D rigid transformation minimising the sum of the squared distances exists and the theta, x0 and z0 values for this transformation are given as below:

$$\theta = \arctan \frac{\sum_k w_k(x_k z'_k - x'_k z_k) - (\sum_k w_k x_k \sum_k w_k z'_k - \sum_k w_k x'_k \sum_k w_k z_k)}{\sum_k w_k(x_k x'_k + z_k z'_k) - (\sum_k w_k x_k \sum_k w_k x'_k + \sum_k w_k z_k \sum_k w_k z'_k)}$$

(Equation 3)

$$x_0 = \sum_k w_k x_k - \cos\theta \sum_k w_k x'_k - \sin\theta \sum_k w_k z'_k$$

(Equation 4)

$$z_0 = \sum_k w_k z_k + \sin\theta \sum_k w_k x'_k - \cos\theta \sum_k w_k z'_k$$

(Equation 5)

Using this distance metric we compute distances between each pair of frames in a pair of motions. This gives us a matrix of distance values with each cell representing the distance between two specific frames within the two motions. On specifying a specific frame pair to be included on a curve we can use dynamic programming to calculate the least cost path to span at least one of the motions. In the application for this project it is desired for looping that the path starts at some matched pair of frames and ends meeting back at those pair of frames. Once the start match frames are imported into the application the matrices are generated so that the start frame matches are at the bottom left and top right corners of the distance matrix and the dynamic programming implementation ensures that the cost path starts and ends at these corners finding the least cost path in between.

As well as the start and end constraints applied to the calculation of the least cost path, three other constraints are imposed. These are as follows:

Continuity -The path must join from the start cell to the end cell without any gaps in the links in between.

Monotonicity - A path is not allowed to go back in time. This is equivalent to not allowing the path to move down or left when traversing from lower left corner at path start to upper right corner at path end.

Slope Limit – At most W_h consecutive horizontal steps may be taken and at most W_v consecutive vertical steps may be taken. The W_h and W_v values are calculated depending on the ratio of number of frames in one motion against the number of frames in the other motion (i.e. The ratio of rows to columns in the distance matrix)

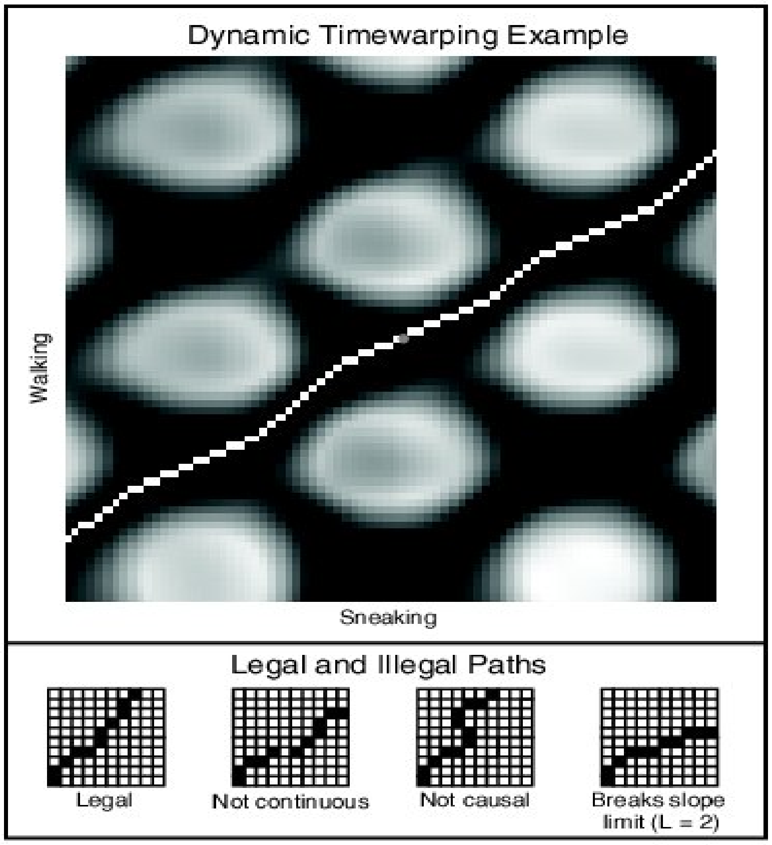

Figure 9: Twisting the shoulder by 30 degrees has a significantly different impact on the posture of the model when the arm is straight (left) and the arm is bent (right).

In the implemented application we allowed three types of steps from a cell on moving toward the end cell, each step can only move one cell horizontally, one cell vertically or one cell diagonally. Other steps (that don't move down or left) could have been added but it was decided to use just those three steps and if required additional steps could have been added at a later date.

The slope limits were defined as:

$$W_h = \min(1.5 * \text{matrixWidth}/\text{matrixHeight}, 2)$$
$$W_v = \min(1.5 * \text{matrixHeight}/\text{matrixWidth}, 2)$$

(Equation 6)

The frame neighbourhood size chosen for the point cloud comparison of frames was chosen as 5 for the application, so that on checking two frames we generate two point clouds with each point cloud generated from the particular frame being checked, its two previous frames and its two next frames. This neighbourhood size incorporates velocity and acceleration into the frame and out of the frame.

Now with a large number of motions to be timewarped together rather than creating a dynamic timewarp curve for each pair of motions which would cause a large number of distance matrices and timewarp curves to be calculated, we use the approximation that if frame Fm1 of motion 1 is matched with frame Fm2 of motion 2, and frame Fm1 of motion 1 is matched with frame Fm3 of motion 3 then frame Fm2 of motion 2 should be approximately matched with frame Fm3 of motion 3. Therefore for n motions say, rather than creating (n-1)! dynamic timewarp curves for each pair of motions we create n-1 timewarp curves by choosing one reference animation (which is chosen as a motion that is most similar to the other motions, e.g. choosing walk rather than an extreme motion such as sprinting). Then we create a dynamic timewarp curve for every other motion in the set paired with this reference animation.

Once all these timewarp curves have been created we fit a clamped uniform B-Spline to each least cost path adjusting the consecutive horizontal and consecutive vertical points on the path to ensure the curve is strictly monotonic.

Following this we sample the reference animation at regular intervals and gather the corresponding times of the other animations from the spline curves and these times are gathered into a single n-dimensional vector (for n motions) to create a single clamped uniform B-Spline curve whose points are n-dimensional vectors specifying the corresponding times for the set of motions. A sampling interval of 5-10 frames of the reference animation for this stage was found to work quite well.

3.2 Alignment and Blending of Motion

In order to create smooth weight adjustment of motions a weight handling class was implemented to maintain and adjust cubic Bezier Splines for the time dependant weight functions of the motions. This manager class could then be used to take in the current timer time and return the motion weights for the animation handling classes to use. These weights are then used in the calculation of the current local to world transforms of the skeletal joints.

The non-root joints were not given degrees of freedom for scale and translate (which were fixed to the bind pose values) so only the joint rotation for the blended/interpolation had to be calculated for each frame update.

With the exception of the root joint the blended/interpolated joint transform relative to it's parent joint coordinate system could be calculated by a weighted averaging of the quaternion orientation of the weighted motions.

The case of the root joint is more complex in that it has more degrees of freedom with 3 DOF for translate and 3 DOF for orientation. Care must be taken in the calculation of the joints new alignment position and orientation in order to correctly produce smooth continuous blending/interpolation of a set of motions.

The majority of the alignment process described in the Registration Curves technique [6] was originally implemented but due to problems arising with implementing the "Positioning and Orienting of Frames" section of this paper, the method was abandoned and a different iterative solution was constructed.

The main process of alignment in the new method was as follows:

1) User/Program specifies where on the x-z plane the projected root joint is to be positioned (i.e. (x, z) world translate) and the y-axis (vertical) orientation of the root in terms of a Euler YZX ordered orientation. The blend/interpolation pose of the first frame of continuous motion is then

calculated with the root position placed at the (x, z) position, the desired root y-orientation set, and the ZX Euler values extracted from the root transforms for each motion and a weighted average of this ZX orientation is then applied to the root positioned and oriented in the y-axis as above. The final root world transform (denote by PrevRootTransform) is stored as is the root world transforms for each motion with a non-zero weight (denote by PrevRootFiTransform).

2) For each consecutive frame the following algorithm is carried out:

```
SET weightedAvgXZTranslation = 0
SET weightedYWorldTranslation = 0
FOR each motion LOOP
    i)   Calculate the current non-aligned root transform for the
         motion. Extract the z-x components of the Euler YZX
         representation of the orientation of the root, and store
         them along with the motion weight for processing after
         loop.

    ii)  Calculate the incremental transform transforming from the
         motion's root transformation stored in previous frame to the
         current root transform of the motion (i.e. $PrevRootFiTransform^{-1}$
         * CurrRootFiTransform), denote this transform by $F_i$.

    iii) Calculate the blended transform of previous frame post-
         multiplied by the incremental transform in step ii) above (i.e.
         PrevRootTransform * $F_i$), denote this transform $M_i$. Store this
         transform and the motion weight for processing after loop.

    iv)  Extract world y translation component from transform $M_i$,
         denote this translation by $Y_i$

    v)   SET  weightedYWorldTranslation =
                    weightedYWorldTranslation + (motion weight * $Y_i$)
    vi) Extract the world x-z translate from transform Mi and subtract
        the extracted world x-z transform from PrevFrameTransform to
        get the world incremental x-z translate.

    vii) Extract the y-rotation transform component of the Euler YZX
        orientation of transform $M_i$.

    viii) Transform the world increment translate in step vi) above by
        the inverse of this y-rotation transform in 5, to get the
        translation local to the coordinate system with y-orientation of
        $M_i$ transformed coordinate system. Denote this localised
        translate by $L_y$.

    ix) SET weightedAvgXZTranslation =
            weightedAvgXZTranslation + (motion weight) * $L_y$

END LOOP

3) Calculate weighted average of $M_i$ orientations stored in step iii) of loop
   and denote by YRotCurr the transform y-rotation of this averaged
   orientation.

4) Calculate weighted average of z-x orientations stored in step i) of loop,
   we denote this transform as ZXRotCurr.

5) Extract (x,z) translation from PrevRootTransform, denote this
   translation transform XZTranslatePrev.

6) Calculate new local to world transform of root as:
      XZTranslatePrev * YRotCurr * weightedAvgXZTranslation *
      weightedYWorldTranslationMatrix * ZXRotCurr
```

Quaternions were used to carry out weighted averaging of the orientations and extracting the Euler YZX components of the orientations.

The reason for transforming the incremental x-z translations to the local space of a coordinate system with y-orientation of the Mi transform was to reduce the effect of collapsing on weighted average of the translations.

For example if we have two motions where the translation from the previous frame of the first motion moves forward and left and the translation from the previous frame in the other motion moves forward and right then averaging the vectors in terms of even the local y-oriented coordinate system of the previous frame will result in a reduction of the translation vector compared to the two vectors that were averaged. By averaging vectors transformed to the y-oriented coordinate system of their respective M_i transform then as the increment in the character's y-orientation is generally in accordance with the incremental x-z translate, the collapsing issue is reduced. Using this technique we would still get collapsing of translations in some cases where translation increment and y-orientation are not in accordance, but this collapse may sometimes be desired, for example with two side-stepping motions where the y-orientation doesn't change, a 50-50 blend would have no translation and no y-orientation change.

For the straight/left/right walk, sneak, limp, and run motions in the dynamic timewarp curve of this application, the above method for calculating the new root position and orientation worked reasonably well (see Chapter 4 for conclusions) in maintaining continuous and correct alignment in the application. Figures 10, 11, and 12 show some frames captured from the application on changing between a sneak and run motion.

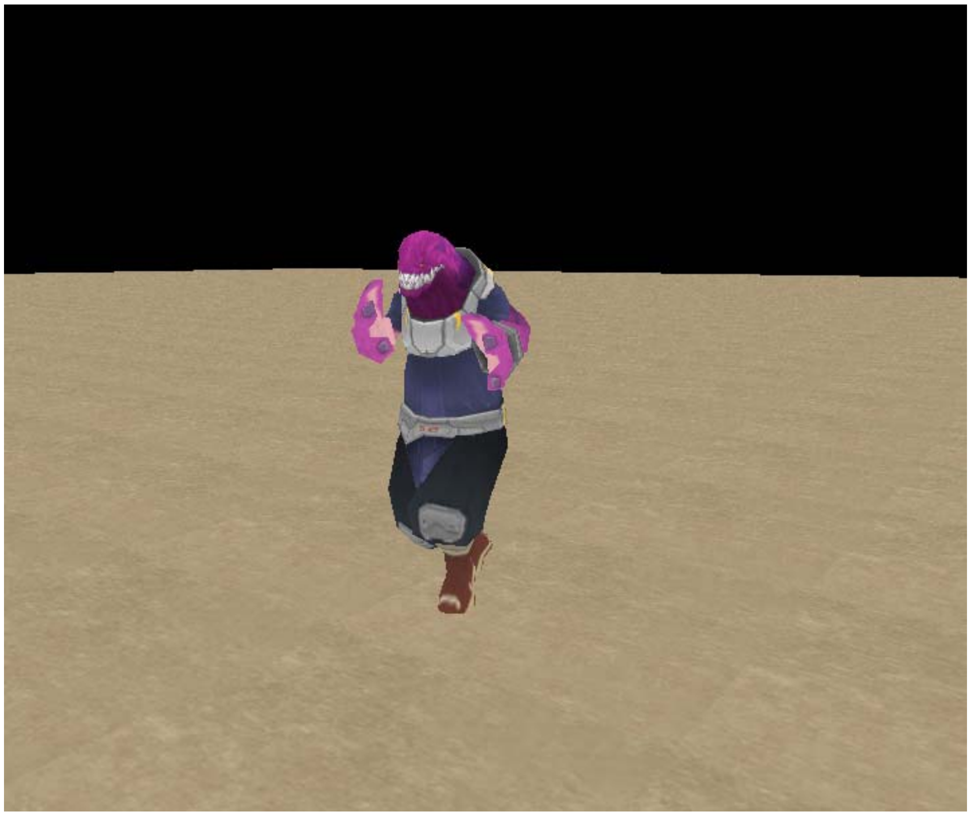

Figure 10: Application playing a running motion cycle of the character.

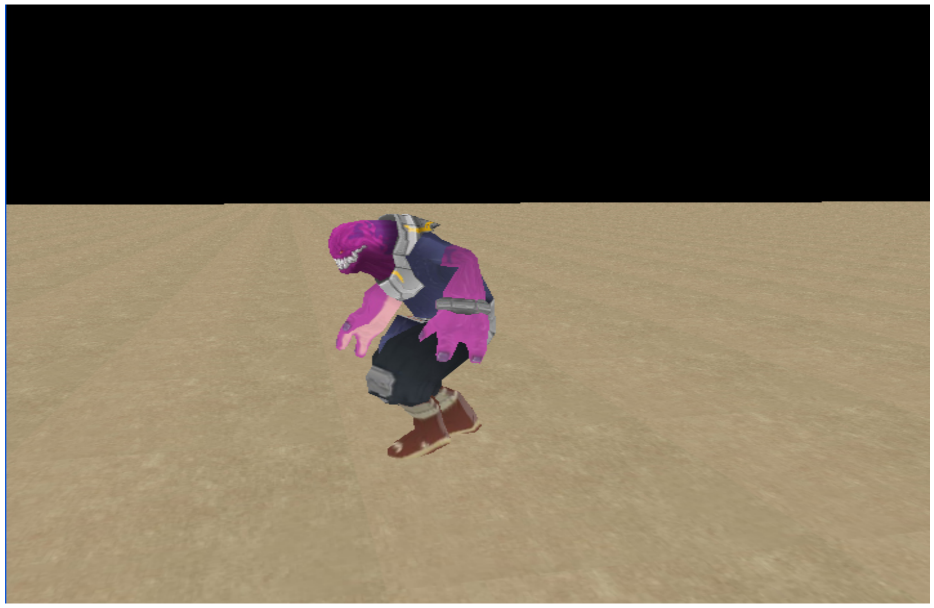

Figure 11: Application playing a sneak motion cycle of the character.

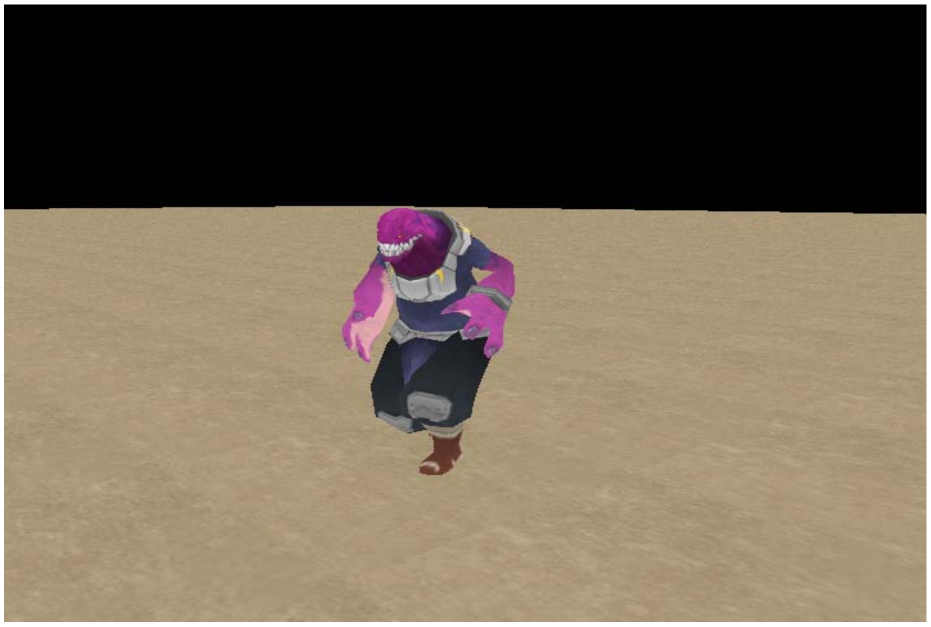

Figure 12: Application playing a more crouched running action on transitioning between run and sneak motion cycles.

3.3 Transitions

In addition to the motions that were grouped for blending/interpolation there were also kick and punch transition motions that could be transitioned to with motion blending.

A finite state machine was implemented to handle the management of transition between blended/interpolated motions and transition motion clips, and to handle the generation of cubic Bezier curves defining the weight transitions between the timewarped blend/transition motion and the non-timewarped transition motion clips. Figures 13 and 14 below show captured frames from the application playing the kick and punch transition motion clips respectively.

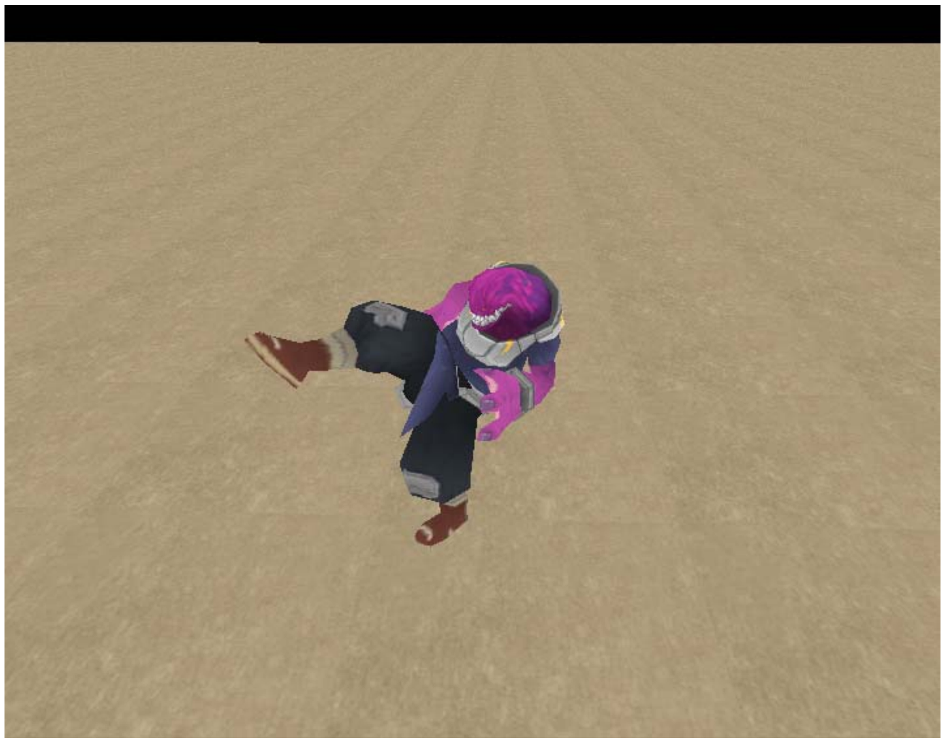

Figure 13: Application running a transition to a karate-kick motion.

Figure 13: Application running a transition to a punching motion.

An XML file containing all the transitional data between these motions such as the frame windows for blending were imported with the transitions to/from the timewarped motion set being in terms of the reference animation. Depending on the current weights of the timewarped motions the duration for transition is altered depending on the speed of the motion relative to the reference animation (for example if the motion has a shorter time span than the reference anim between matched frames such as a run compared to a walk then the transition duration was lengthened). A simplified linear blending method was used for calculating the time increments along the transition motion clip animations, with the time scaling factor based on the current motion weight and the ratio between the actual transition time and the transition motion frame window duration.

CHAPTER 4: CONCLUSIONS

4.1 Results and Future Work

The project did achieve the main goal of providing interactive streams of motion. Unfortunately there was not time to apply constraints to the motion due to an underestimation of the problems that would be encountered during the implementation of the application.

There are many areas where the application could be improved. The most immediate improvement being to fix a bug which seems to be in the finite state machine logic causing a time-blip on transitioning back from a transition clip to the previously running blended motion. This glitch causes the character to rotate a large amount in one frame and motion is continued from the newly oriented position.

The other main bug for which there was no time remaining to debug was in the weight adjustment for weight interpolation and this aspect was removed prior to the hand-in, leaving just the transitions within the timewarp-matched animations and the transitions to the non-timewarped transition clips (repeated timewarped motion transition requests during transition phase of timewarped motions does play motion with more than two non-zero motion blending weights so it may be a trivial fix to the application).

It would also be beneficial to tidy up a couple of the animations which have a slight discontinuity on the re-looping, some of the animations were fixed but the turning run motions in particular require adjustment.

One other issue with the turning run animations regards the non-uniform turning motion which resulted in matched phases of opposing run turns having relative differences in banking and rate of turning (as the animations are matched with corresponding leg movement). These differences caused variable acceleration of banking and rate of turning on transitioning between two opposing turns of the run motion.

The user interface is also an area which can be improved upon, a graphical user interface would be ideal for adjusting the weights with sliders to produce interpolated sequences of motion.

It would also be interesting to see how the application works with more uncommon motions, say a character following curved paths with variable spinning orientations, moving backwards and so on and see how the motions blend together.

In terms of the addition of applying constraints it would be interesting to have the application adjusting actions such as kicks and punches to particular targets and using a footplant constraint solver such as Kovar et al's [6] system and attempt to apply the captured motion to uneven terrain.

4.2 ACKNOWLEDGEMENTS

The author would finally like to thank Andy Cousins of Access Mocap, Bournemouth University in addition to James Whitworth and Jonathan Macey for their advice and guidance throughout the project.

References

[1] Kovar L., Gleicher M., and Pighin F., *Motion Graphs*, ACM International Conference on Computer Graphics and Interactive Techniques 2002, p473-482.

[2] Kovar L., Gleicher M., and Shreiner J., *Simulation, Motion Capture, Editing : Footskate Cleanup for Motion Capture Editing*. ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2002, p97-104.

[3] Gleicher M., Shin H. J., Kovar L., and Jepson A., *Snap-Together Motion : Assembling Run-Time Animations*, ACM Symposium on Interactive 3D Graphics 2003, p181-188.

[4] Menache A., *Understanding Motion Capture for Computer Animation and Video Games*, Morgan Kaufmann Publishers Inc, 1999.

[5] Ikemoto L. and Forsyth D. A., *Enriching a Motion Collection by Transplanting Limbs*, ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2004, p99-108.

[6] Kovar L and Gleicher M., *Flexible Automatic Motion Blending with Registration Curves*. ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2003, p214-224.

[7] Tolani D., Goswami A. and Badler N. I., *Real-time inverse kinematics techniques for anthropomorphic limbs*. Graphical Models 62, 2000, p353-388.

[8] Wu X., Ma L., Chen Z., Gao Y., *A 12-DOF Analytic Inverse Kinematics Solver for Human Motion Control*. Journal of Information and Computational Science 2004, p137-141.

[9] Lee J. and Shin S. Y., *A Hierarchical Approach to Interactive Motion Editing for Human-Like Figures*, ACM International Conference on Computer Graphics and Interactive Techniques 1999, p39-48.

[10] Shin H. J., Lee J., Gleicher M., and Shin S. Y., *Computer Puppetry: An Importance Based Approach*, ACM Transactions on Graphics (TOG) 2001, p67-94.

[11] Watt A., Watt M, *Advanced Animation and Rendering Techniques*, Addison Wesley. 1992

[12] Welman C., *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation*, Masters Thesis, Simon Fraser University. 1993

[13] Buss S. R., *Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods*. Unpublished survey article. 2004.
Available at
http://www.math.ucsd.edu/~sbuss/ResearchWeb/ikmethods
(Accessed 03-Aug-2006).

[14] Buss S. R. and Kim J. S., *Selectively Damped Least Squares for Inverse Kinematics*, Journal of Graphics Tools, 2005, vol. 10, no. 3, p37-49.
Available at:
http://www.math.ucsd.edu/~sbuss/ResearchWeb/ikmethods
(Accessed 03-Aug-2006).

[15] Gleicher M., *Comparing Constraint-Based Motion Editing Methods*, Graphical Models, 2001, 63(2):p107-134.

[16] Gleicher M., *Retargetting Motion to New Characters*, ACM International Conference on Computer Graphics and Interactive Techniques 1998, p33-42.

[17] More A, and Gleicher M., *Building Efficient, Accurate Character Skins from Examples*. Proceedings of ACM SIGGRAPH 2003, p562-568.

[18] Lewis J. P., Cordner M., and Fong N., *Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation*. ACM International Conference on Computer Graphics and Interactive Techniques 2000, p165-172.

[19] Kry P. G., James D. L., Pai D. K., *EigenSkin: Real Time Large Deformation Character Skinning in Hardware*. ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2002, p153-159.

[20] Kranf Site : Frank Vanden Berghen C++ XML Parser. Available at: http://www.applied-mathematics.net/tools/xmlParser.html (Accessed 10-Sep-2006).