Motion Blending

Kristine Slot

February 2007

Abstract

When constructing an animation of a character, it is of interest to make this character able to perform different motions. Thus it is of interest to find a way to change from one motion to another, without it being an abrupt change or a change between two positions that have no similarities. This can require a lot of manual work from an animator and different settings for different animations.
This project makes a generic approach for blending two motions. We use timewarping to automatically determine the timing between the two motions and apply alignment of the motion to control the orientation of the result motion.
The goal of this project is to make a change between two motions seem as realistic as possible and doing so without any work from an animator.

Contents

1 Introduction **4**

2 Character **5**

3 Character motions **6**

4 Motion Blending **8**
4.1 Motion blending 8
4.2 Motion transition 8

5 Related work **9**

6 Linear Blending **11**

7 Timewarping **12**
7.1 Finding the shortest path 14
7.1.1 Finding the starting point 16
7.1.2 Ending the path 17
7.1.3 Representation of the path 17
7.2 Using the path 17

8 Moving the character **19**

9 Alignment Curve **22**

10 Implementation **24**
10.1 Sequence class 24
10.2 Blender class 24
10.3 Graph Blend Schedular class 25
10.4 Other files 25

11 User guide **26**

12 Testing **27**
12.1 Distance maps 27
12.2 Timewarping path vs. linear blending 29
12.3 Changing time step 35
12.4 Alignment 36
12.5 Root motion 36
12.5.1 Original motions 36
12.5.2 Blended motions 36
12.6 Blending with partial motions. 37

13 Conclusion **38**

1 Introduction

This project is written by Kristine Slot, at Copenhagen University, Department of Computer Science. It is assumed that the reader has some understanding of computer science, graphics/animation terms and mathematics used in graphics. The paper will run through some of the background theory concerning the main areas of this project.

This project is constructed to be a part of the OpenTissue project [10]. In this it is only possible to create a linear blend between motions. When not handling the blend, the result can be unrealistic due to no control of starting the blending when motions are similar and not considering of the animation.

To solve this problem I wish to create a generic approach for performing the linear blend. This project is concerning the development of registration curves, which in three parts genericly chooses how to perform the linear blend.

Timewarping, where we find the optimal place for starting a blend and then performing the blend by timing the two animations so the frames being blended continues being good matches.

Alignment curves considers the problems when blending two animations with an angle larger than 180 degrees. If we use a normal linear blend, the blended animation will turn in an undesired direction once the angle passes 180 degrees.

Constraint Matching where we consider the problem that since we are combining the frame by finding the average, some constraints may fail. Hence the constraint matches are created containing a set of related constraints for each motion. The start and the end of these constraints in global time are treated as parameters which may themselves be blended.

This project will make a registration curve with timewarping and alignment curves.

The implementation of the registration curves will be developed to be part of the OpenTissue project and as such a lot of components for making an animation will be used from this.

2 Character

We wish to represent a character in a 3D environment. For doing this, we use a right-handed coordinate system. This consists of the three axes x, y and z, where the x- and y-axes define the floor-plane, and the z axis defines the up-going vector. In figure 1, an example is shown where a character is placed in such a coordinate system.

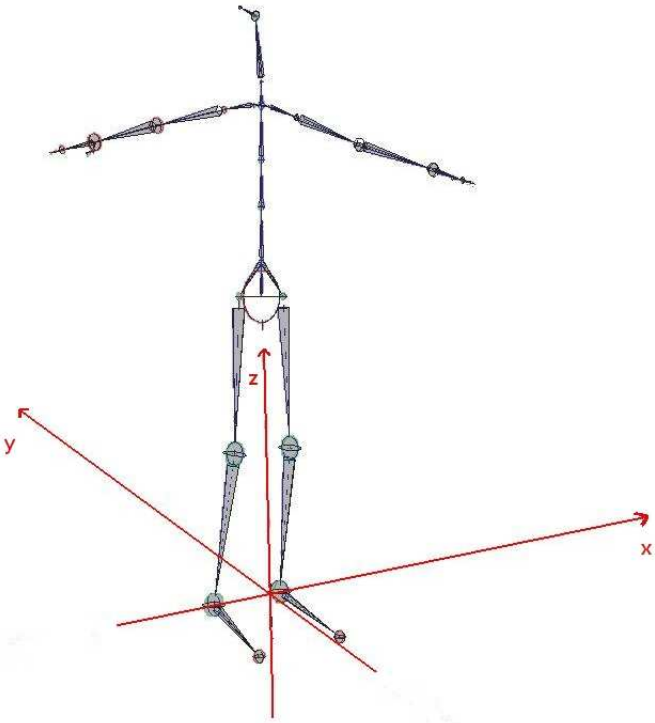

Figure 1: A character standing upright and the red lines defining the coordinate system.

A character is represented as a skeleton, consisting of bones, which are being held together by joints. A joint is merely what represents the character. By giving a joint a position, a rotation and information about to which joint it is connected (its parent joint), it is possible to build an entire character. In figure 2, we see two bones connected by a joint, and the parent of the joint is the joint connected to the upper bone.
By changing the position of a joint it is possible to move the character, and by changing a rotation it is possible to change the angle between two bones (eg. like when bending a knee). In figure 3, we se a translated and a rotated joint, and we see the effect of the bones surrounding the joint.

As mentioned above, a joint has a parent. Which means, that a joint is able to have one or more children. A joint i is child of a joint j if j is the parent of i.
When changing the position or rotation of a joint, it does not only have an effect on the

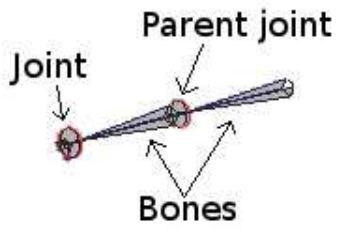

Figure 2: Two bones joint together by joints, where one joint is the parent of the other.

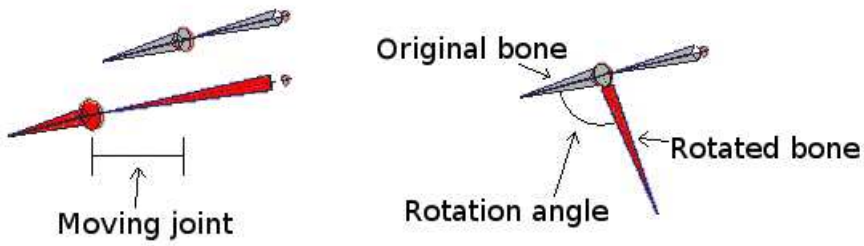

Figure 3: The effect of moving and rotating a joint (respectively left and right picture). The red bones represent the bones after the joints having been moved or rotated.

selected joint, but also on all of its children. This can be seen in figure 4, where a joint is rotated and the children follow this rotation.

In order for every joint to be connected to a parent, one joint must hold the property of being the root joint. This joint is the first set, and following the path of any joint in the character, to the parent through every bone, all end up in the root joint.
In this project we define the root as being the pelvis of the character. Since every child moves according to its parent, a transformation of the root will lead to a transformation of the whole character, as seen in figure 5

3 Character motions

As shown in section 2, the use of joints and transformation of these, enables us to create a character in motion.

Given a character, with a skeleton consisting of N joints, a motion is defined as several sets of N joint transformations, where each set is associated with a unique increasing time value.

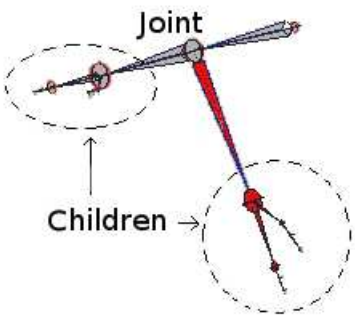

Figure 4: An arm joint is rotated, and the children of this joint rotate along with the joint. The red arm represents the rotated arm

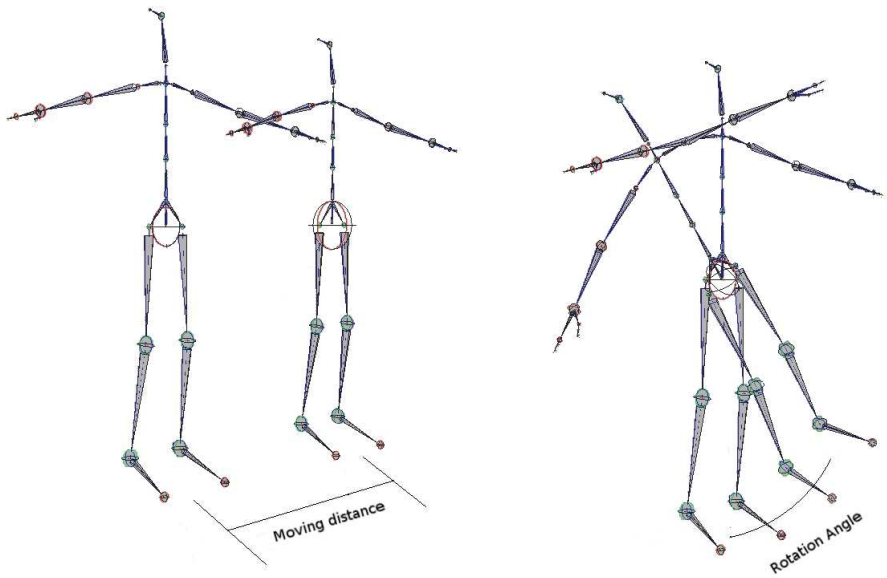

Figure 5: It is seen how the transformation of the root joint changes the whole character

Basically we think of two different kinds of motions. The cyclic and the noncyclic motion. A cyclic motion can be repeated and still look plausible, hence the end of the motion must be identical to the start of the motion. An example of a cyclic motion would be running or walking. By making these cyclic, it is possible to continue the motion, and thus save a lot of memory by only having to save one cycle and not a long animation.

A noncyclic motion is only meant to be played once. It is then not necessary for the start

and end joints to have the same values. It would not look as a plausible motion if we did play it again, since there would be foot-skating or jumps in the animation

4 Motion Blending

When having defined different motions for the character to perform, we want to be able to make a transition from one motion to another, or to make a combination of motions by blending two motions together.

4.1 Motion blending

By blending several motions, we obtain the possiblity of doing multiple motions simultaneously. This means that while blending, the final motion is a blend of all the active motions. Each frame in the final motion is found by blending each joint of the character with the corresponding joints in the other motions (eg. elbows are blended with elbows, etc).
Two joints are blended by blending rotation and position of the joints by using weights to determine how the joints should be weighted in mutual dependence.
Methods for constructing a blend, are discussed in the following section, where we look at theories on making a motion blend, and in the rest of this report on making our own motion blend.

4.2 Motion transition

A motion transition also uses blending. The difference lies in the fact that a transition is a temporary blend. A transition will usually be used for changing from one motion to another. When making the decision of changing from one motion to another, a motion blend is started, but only for a certain transition length. This length should be determined from case to case depending on the blended motions. The weight mentioned above is modified so that the starting motion has the highest weight when starting the transition, and the ending motion has the highest weight at the end of the transition. This is termed a cross-fade.
These weights are determined from the transition length, and thus when the transition has run the desired length, the ending transition has a weight of one and the starting transition of zero. This means that we have changed from one motion to another.

5 Related work

This project concerns the problem of motion blending and transitions and making these look as realistic as possible. By this we mean, that the final animation of a blend or a transition should not have any jumps in the animation, due to bad chosen animations to be blended. It is also desired to do everything automatically and avoid manual user intervention. Before doing this we take a look at previous methods and the development of the process of motion blending.

Making a motion transition or a motion blend between two motions has been a known problem with various authors solving it in different ways. Perlin [11] presented a real-time system where he created motions based on a noise function, and thereby creating realistic motions.

Bruderlin and Willams [1] used signal processing for making blended transitions of motions, by introducing multi-resolution motion filtering to be applied to existing motion data. By adjusting gains of bands for joint angles they made it possible to create reliable blended transitions. This method still suffered from being non-automatic and only working with two motions.

Witkin and Popović [15] introduced another method of blending motions by using motion parameter curves. The animator would have to define a set of keyframe constraints to smoothly add small changes to a motion, when motion clips were found, by overlapping and blending the parameter curve. Still this method required a lot of work for an animator.

A method for automatically blending motions was introduced by Kovar et al. [8] in the form of motion graphs. The strategy of the motion graph method was to automatically identify a portion of motion frames that were sufficiently similar, so that straightforward blending was almost certain to produce valid transitions. These clips were found by using a calculated distance between each frame from both motions, hereby creating a matrix, containing the distances, from which an optimal path could be created to find the timewarping between the motions. This idea was originally introduced by Schödl et al. [12] in an article concerning video textures and how to edit video sequences by removing or adding sequences in the middle of a sequence.

Simultaneously with the development of the motion graph method, a similar method was developed by Lee et al. [9]. This method also used graph structure to create a plausible transition between motion segments.
They created three user interfaces: a list of choice, a sketch-based path through an environment similar to the one used in motion graphs, and live video feed.

Gleicher et al. [4] developed the timewarping method further by also considering the neighbours of the frames for which the distances were to be computed. Then it was possible to create two point-clouds, one for each motion, containing joints from the frame and its neighbours. By aligning them and finding the distance for each corresponding point across the two motions, they found a better total distance, which did not only look at one frame, but also considered if the frames were alike before and after, eg. the same arm is moving

forward in both motions around these certain frames.

Later on this method was further developed by Kovar and Gleicher [7], when they introduced the concept of registration curves. This concept was not only using timewarping and creating a timewarping curve, but also using coordinate frame alignment, which also considers the case where the linear interpolation could fail. This could happen if the two motions had different directions while they were blended. Since the root position is linearly interpolated, it would cause a collapse of the root paths and the character would flip around at the end of the motion. Each frame that is to be blended has an influence on the position and orientation, and the influences are combined according to the blend weights. The registration curve creates an alignment curve which aligns the frames for each point on the timewarp curve.
The article also added constraint matching to the registration curve. The constraint matches solve the problem that arises when the frames are combined averagely. Then the constraints on the blend may not be fulfilled. The constraint matches are created containing a set of related constraints for each motion. The start and the end of these constraints in global time are treated as parameters which may themselves be blended.

For all systems working with transitions a certain blend length will be used. Wang and Bodenheimer [14] introduced a way of finding the optimal blend length for a transition between motions.

Registration curves have been used in further development over the years. Heck et al. [5] developed a method for splicing an upper-body motion with a lower-body motion and thereby creating an automatic method for making a character perform different motions simultaneously, like walking while carrying a box. When blending two motions, the article uses time warping as in [8, 7, 4] to make sure the motions for the upper body is in the same phase as the locomotion cycle for the lower body. Then the upper body is rotated to align the pelvis with the lower body.

Another example of the use of registration curves is the article by Kovar and Gleicher [6], concerning identifying logically similar motions from a large data set. This means finding multiple animations performing the same motion in different variations, eg. two different kicks. The problem considered in this article being that two motions such as two kicks (that should be recognized as the same motion) are not necessarily similar all the way. They find numerically equal points for small segments by using a time warping structure, and create a match web of possibly similar motion segments. To recognize related events, additional functionality is added to the timewarping method, by finding all local minima in the matrix. Then it is possible to further recognize if a motion is similar to another.
A problem with this method is that it is still difficult to sort out the motions that do not perform the same action, but are very similar.

6 Linear Blending

As seen in the previous work (section 5) linear blending is being used in every article. Thus this section describes this sort of motion blending.

As mentioned in section 4, we wish to blend the corresponding joints of the motions. If given two motions, $\mathbf{M}_i$ and $\mathbf{M}_j$, which are to be blended, one with a weight w_i and the other with a weight w_j, then the blended position of the k'th joint of the two motions ($\mathbf{J}_{i_k}$ and $\mathbf{J}_{j_k}$) are found by

$$\mathbf{J}_{k_{pos}} = \frac{\mathbf{J}_{i_k} \cdot w_i + \mathbf{J}_{j_k} \cdot w_j}{w_i + w_j}.$$

Equally for N animations blended together the position of the k'th joint can be found by

$$\mathbf{J}_{k_{pos}} = \frac{\sum_{i=0}^{N} \mathbf{J}_{i_k} \cdot w_i}{\sum_{i=0}^{N} w_i}.$$

The rotation of a joint is represented by quaternions, hence it is possible to construct the linear blending by using spherical linear interpolation as introduced in [13]. By having a starting quaternion q_s and a quaternion q_b to be blended with q_s, we can find the linear interpolation q_l as

$$q_l = q_s(q_s^{-1} q_b)^{w_b}, \tag{1}$$

where w_b is the weight between 0 and 1 that determines how much q_b should be weighted according to q_s.
For blending more than two quaternions, we start by finding q_l for the first two quaternions and then setting $q_s = q_l$ and performing equation 1 for the next quaternion. This continues until all quaternions has been blended.

A blend is seen in figure 6 where the grey and the red legs are two motions being blended and the green legs are the blended legs.

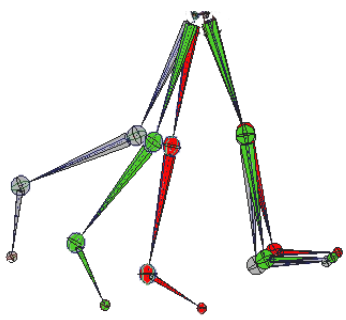

Figure 6: A linear blend of two motions. The grey and red legs are the original motions, and the green legs are the blend between them.

7 Timewarping

The linear blending described in section 6 implies two serious problems. It does not consider the best position to construct the blend, and it does not consider the problems if one motion is changing faster than the other. So even though we find a good place to begin the blending in form of similar points, we can still have a strange blending while using a normal linear blending.
This section will introduce a method, by which the input to a linear blender can be controlled, and thereby making sure that the frames being blended by the linear blender are as similar as possible. This will make the blending look more plausible.

We use a method called a timewarp curve, which was first introduces by Kovar et al. in [8]. The timewarp curve finds the best path of frame pairs (one from each motion) through the two motions. To find these frame pairs, it is of interest to find a way to determine how well two frames match. In other words, we are interested in comparing the corresponding joints and from the distance between all the pairs of joints determine how good two frames match.
To find these matches, a map has to be created, containing the distances of each pair of frames. It is necessary to calculate the distance between each frame of the blending motions, thereby creating a $N \times M$ map, where M is the number of frames in the source motion and N is the number of frames in the destination motion. For each cell (i, j) in the map, the difference between the i'th frame in the destination motion ($\mathbf{F}_{dst}(i)$) and the j'th frame of the source motion ($\mathbf{F}_{src}(j)$) are computed.
The distance $\mathbf{D}$ in a point (i, j) in the map, are found by calculating the distance between each joint in the frame $\mathbf{F}_{src}(j)$ and the frame $\mathbf{F}_{dst}(i)$ and then finding the weighted sum of the differences.

Given two frames for which we are to find the distance (see figure 7,a) we need only to look at the joints, so we divide the two frames into points as seen in figure 7,b. A point is defined as the origin of a joint. Thus creating two point-clouds (one for each frame) and a total point cloud consists of every joint in a frame, being represented as a point.
Finding the weighted distance for corresponding points in the two point-clouds can be thought of as least square fitting, where we wish to minimize the distance

$$\mathbf{D}(\mathbf{F}_{dst}(i), \mathbf{F}_{src}(j)) = \sum_{k=1}^{n} ||\mathbf{p}_{k,j} - \mathbf{p}_{k,i}||^2,$$

where $\mathbf{p}_{k,j}$ is the kth point of the jth frame of the source animation. To minimize this distance we wish to make a least square fit on the destination animation points ($\mathbf{p}_{k,i}$). Thus the destination frame should be translated into the source frames coordinate system. We rotate about the vertical axis (the z-axis), because a walking motion for example looks the same whether it is started at the origin or at any other place on the floor-plane. Thus we can conclude that a motion is fundamentally unchanged about this axis, and translate along the floor (x, y) plane. This implies that we can apply a 2D transformation to the destination points to change the local coordinate system to the coordinate-system of the source points. This transformation is seen in figure 7, c. When the points are transformed, it is possible to calculate the distance between corresponding joints

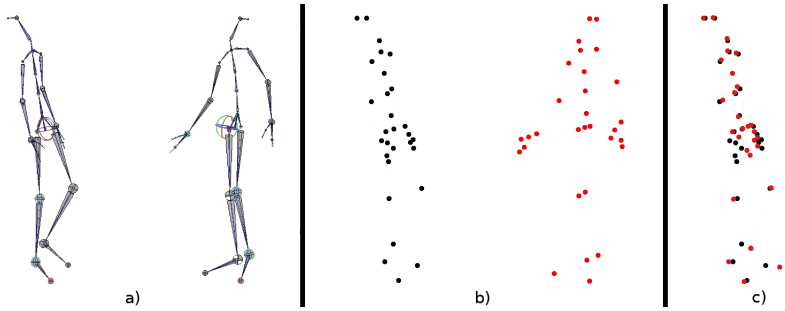

Figure 7: The transformations of one frame into the other. Step a) shows the two frames. Step b) shows how we only work on the points of the skeleton and step c) shows the transformation of the red character onto the blacks character's coordinate system

A transformation is found for every pair of frames (i, j). In order to find this we have to find the mean position for each frame. The mean position is denoted $\bar{p}$ and is constructed from the n points in the skeleton in that particular frame. A mean value is calculated as

$$\bar{p} = \sum_{k=1}^{n} w_k p_k, \tag{2}$$

where p_k is the kth point in the skeleton and the weight w_k is used for giving more importance to certain points. The sum off all w_k's sum to unity.

Then the rotation (θ) about the z-axis and the translation coordinates x_0 and y_0 can be found as the optimization ensuring a least square fit:

$$\theta = \tan^{-1}\left(\frac{\sum_k w_k(x_k y'_k - x'_k y_k) - (\bar{x}\bar{y}' - \bar{x}'\bar{y})}{\sum_k w_k(x_k x'_k + y_k y'_k) - (\bar{x}\bar{x}' + \bar{y}\bar{y}')}\right) \tag{3}$$

$$x_0 = \bar{x} - \bar{x}'\cos(\theta) - \bar{y}'\sin(\theta) \tag{4}$$

$$y_0 = \bar{y} + \bar{x}'\sin(\theta) - \bar{y}'\cos(\theta), \tag{5}$$

where $\bar{x}$ and $\bar{y}$ is found from equation (2), where $\bar{p} = (\bar{x}, \bar{y})$.

When the transformation has been found, it is possible to find the distance between two frames.

The distance between two frames, $\mathbf{F}_{dst}(i)$ and $\mathbf{F}_{src}(j)$, is found as the minimal weighted sum, as defined above, and is denoted $\mathbf{D}(\mathbf{F}_{dst}(i), \mathbf{F}_{src}(j))$ and defined as

$$\mathbf{D}(\mathbf{F}_{dst}(i), \mathbf{F}_{src}(j)) = \min_{\theta, x_0, y_0} \sum_{k=1}^{n} w_k ||\mathbf{p}_{k,j} - \mathbf{T}_{\theta,x_0,y_0}\mathbf{p}_{k,i}||^2,$$

where n is the number of joins, $\mathbf{p}_{k,i}$ and $\mathbf{p}'_{k,j}$ are the kth point of the two frames, and $\mathbf{T}_{\theta,x_0,y_0}$ is the linear transformation. Having w_k summing to unity, $\mathbf{D}(\mathbf{F}_{dst}(i), \mathbf{F}_{src}(j))$ is the weighted

sum of the distances from each pairs of joints.

We are now considering the fact that, even though two positions in the blending motions have similar poses, they are not necessarily a good match if they act differently in the nearby frames, eg. due to different velocities, accelerations or directions. From this fact emerges a wish of also adding the neighbourhood frames when calculating a distance. It is possible to give the neighbor frames a lower weight (w_k) to ensure that the current frame is still being considered the most important. This way of solving the problem can be compared to the finite difference approach for approximating a curve. For each neighbour included we add a new derivative to the finite difference, hence considering velocity by adding one neighbour on each side and by adding two neighbours we consider the acceleration.

We imagine creating two point-clouds. One for the source and one for the destination motion (the black and red clouds in figure 7, b)). Each point in one point-cloud corresponds with a point in the other cloud. The final distance will be the sum of comparisons between all point correspondences. The weights w_k must still sum to unity.

We define X as the number of neighbors used in the distance calculation. Then the calculation should be performed for $2X+1$ frames (the current frame plus X neighbor frames on either side). To make sure that the current frame gets a weight higher than its neighbors, we set a weight w_k for each frame we compute, where the current frame has the highest weight, the closest neighbor the second highest and so on.
We start by assigning a weight to each neighbor group. If we define $m \in [-X; X]$, where $m = 0$ is the current frame group, we find w_m as

$$w_m = \frac{X + 1 - |m|}{(X+1)^2},$$

where $|m|$ is the absolute value of m.
The weight w_k will now be divided by the number of points in the skeleton to ensure that the sum of w_k for all points is one.

$$w_k = \frac{w_m}{point\_count}.$$

A calculated distance map will look something like what is seen in figure 8, where the black areas are the small distances and the white the large ones. It is already possible to see an optimal minimum distance path being computed along the diagonal.

7.1 Finding the shortest path

When the distance map has been computed, it is possible to find the shortest path through the distance map. We denote this path as the curve $\mathbf{S}(u)$, and it returns, for a given local time u, which source frame should be blended with which destination frame. To ensure forward motion on both animations, we accept no holes in the curve and no still standing motion at either side. We define three conditions which must be fulfilled.

Continuity The path must be continuous. Meaning, each cell on the path must share a corner or an edge with another cell on the path.

Causality The path must not reverse direction, because then the animation will reverse.

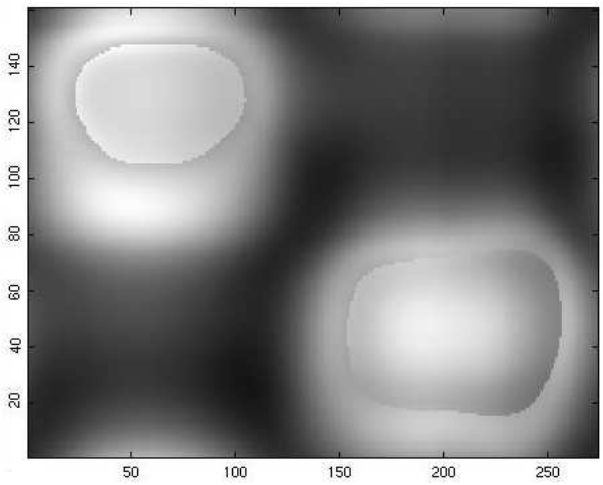

Figure 8: An example of a distance map, where the dark areas represent a small distance and the while areas a large distance.

Slope limit A given slope limit L must be fulfilled, so that no path takes more horizontal or vertical steps than L.

We now want to determine a method of finding the points in the distance map that should define the path curve $\mathbf{S}(u)$

Given a part of a shortest path, and wanting to find the next point, we choose the next valid point that has the smallest distance value.
A valid point can be defined by considering the above mentioned conditions.
To ensure continuity we always choose one of the nearby frames to the last frame found on the path. As an addition to this we ensure causality by only having three nearby frames as the next possible choice to the path. In figure 9 the black point (i, j) in the array, is the last found point on the path. The three grey points, $(i, j+1), (i+1, j)$ and $(i+1, j+1)$ are the valid candidates for the next point in the path.

The slope limit is ensured by having a counter on how many consecutive times the vertical or the horizontal points have been chosen. If this exceeds the slope limit, the critical point is not even considered, but we only consider the two remaining. This can be seen in figure 10 where the horizontal axis has reached the slope limit and therefore only has two valid points to choose between. In this case the vertical point and the corner point. When finding the vertical point to be the smallest distance of the two and thus adding this to the path, the horizontal counter is set to zero, but the vertical counter is increased. But it has not yet reached the slope limit and therefore all three points are valid points for the next selection.

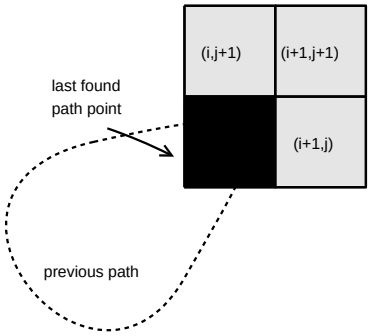

Figure 9: The black area shows the last point of the chosen path and the grey area represents the three valid choices.

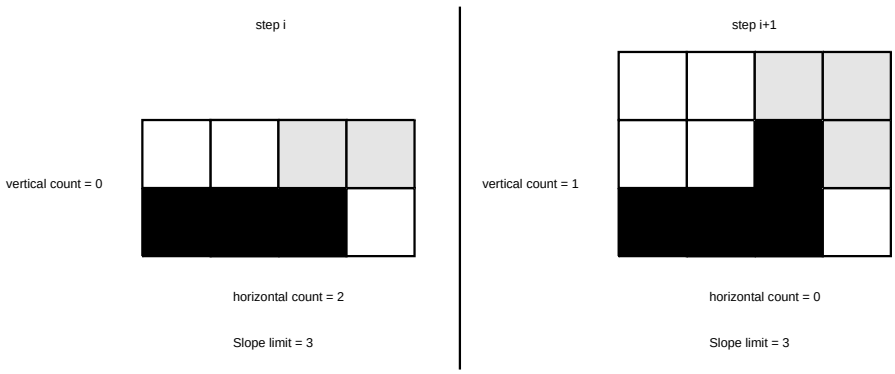

Figure 10: In step i the slope limit of 3 is reach in the vertical count and thus we only have two valid points for the next point in the path (the grey boxes). Next in step $i + 1$ we again have all three choices.

7.1.1 Finding the starting point

Until now we have considered how to find the next point in the path assuming that a previous point has already been discovered. Now we need to consider how to find the first point.
The first point must lie on either the beginning of the destination axis or on the beginning of the source axis. Meaning on either $(0, y)$ or $(x, 0)$ where $x \in [0; N]$ and $y \in [0; M]$, where we remember from previous in this section that M is the number of frames in the destination motion and N is the number of frames in the source motion. If the destination motion is noncyclic, it is important that the starting point has $y = 0$, since it will look odd to start in the middle of a motion.
The point with the smallest distance among these choices is not necessarily the best choice for starting the path. It is possible that the values after this point have large distances. Therefore the only way to find the best path is to try calculating the path for every possible starting point and summing the distances. The path with the best summed distance is the optimal minimum distance path.

7.1.2 Ending the path

The length of the path should cover every frame of the destination motion. This is also a reason for starting a noncyclic destination motion in $y = 0$. Otherwise the path cannot be looped if reaching the end of the map, as it should be if a motion were cyclic.

In figure 8, we showed the distance map. The computed path using the algorithms described and a slope limit of three, can be seen in figure 11.

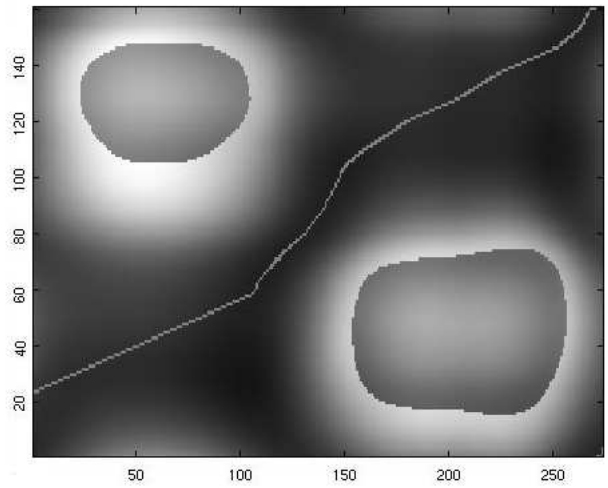

Figure 11: An example of a distance map with the corresponding optimal path. The slope limit is set to three

7.1.3 Representation of the path

We want to represent the path $\mathbf{S}(u)$ in a cheap easily accessible way. In [7] the path $\mathbf{S}(u)$ is approximated as a spline. This gives a good approximation if the sample rate is larger than the sample time of the animation. But if we have a high sample rate, which is necessary for making a reliable distance map, it is easier to represent the path by an index to the memory. By a simple lookup in the path by a timestep u we can easily and cheap find the next index of the path.

A memory block consists of an index (i, j) to the distancemap to determine the timewarping and angle θ and the translation coordinates (x_0, y_0) to determine the alignment (described later in section 9)

7.2 Using the path

When the list of integer pairs have been found, we can execute the timewarping. This is done by finding a starting point. A starting point can be determined depending on whether the destination motion is a cyclic or noncyclic motion (se section 3). If the destination motion is noncyclic, it will look odd to start the blending in a point that has an index number in the

middle of the motion. Therefore the source motion should run until it reaches a match with the beginning of the destination motion.
If the destination motion is a cyclic motion it does not matter where we start the motion, and thus it can be started as soon as we reach a source index included in the path.

When an integer pair has been chosen we perform a linear blend between the two frames.

8 Moving the character

Real motion captured data often include root motion. When using an animation like in this project, where we hand-craft a motion to be continued if it is cyclic, the root motion is not included.
Moving the root of a character moves the whole character as shown in figure 5 in section 2.
The easiest way to move the character is to use an ad-hoc estimation of the motion, but this can result in foot-skating. To avoid this, we choose moving the character as its feet demand. If we imagine a person doing a walking motion, the feet are determining how we move forward. A foot planted in the ground will be standing still, while the rest of the body moves. In the case of the static motion, the pelvis is the fixed point, and the feet are not kept static on the ground. By changing this so that the pelvis moves according to the feet, where the demand is that the feet should be kept static, it is possible to create a valid motion forward.
Since the motion are already moving in the z-direction (up and down), we only concern our self on moving the character along the x- and y-axis.

By adding a plane on the z-axis, we define a foot as being on the ground when it passes this plane (see figure 12). By aligning two frames, between which we would like to move the

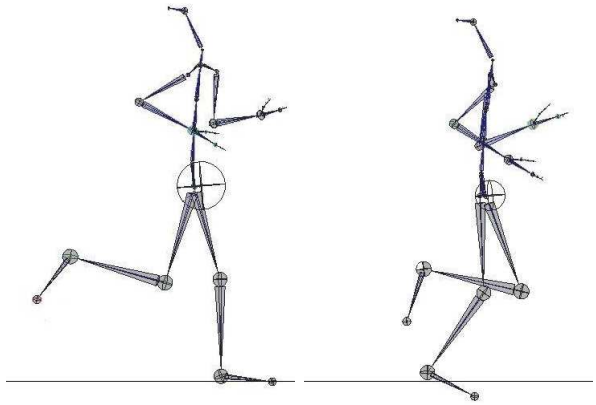

Figure 12: Two motion positions where the character is putting a foot trough the ground. Hence the foot should be static from where it first hits the plane.

character, we align the roots of the two frames. For a correctly made motion this should be the same as aligning the feet to the same plane on the z-axis. This can be seen in figure 13, where the grey character is the first frame in which the foot steps onto the plane, and the red character is the next frame which should be made moving according to the foot position of the first frame.
By moving the pelvis as far as the feet have been moved (in both the x- and y-direction, the feet should remain at the same spot for as long as the feet are on or below the plane. This will create a reliable forward moving motion. See figure 14.

A problem arises when a running motion occurs. A running is characterized by having series of time where no feet are touching the ground. This problem is solved by saving a velocity from the previous calculation between two motions, both having a foot on the

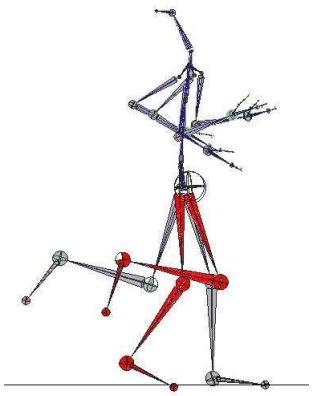

Figure 13: A motion being moved forward by aligning two frames of this motion. The grey character is the first frame, and the red the second

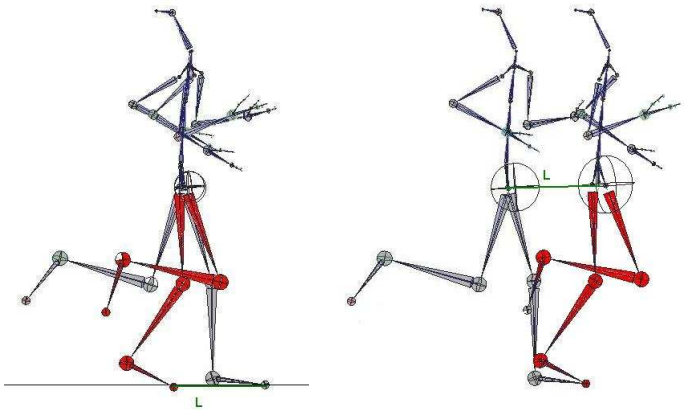

Figure 14: A static character coming into forward motion by moving the pelvis in the (x, y) coordinates that the foot on the plane moved, thereby creating a static foot and a forward motion.

ground. In sections where no feet are touching the ground, we will need to calculate the position by using a saved velocity and angle to the floor plane, from the last two frames where the character has a foot in the ground.
By adding physics laws it is possible to determine the velocity for each time step where the character is in free-fall. The gravity, g will keep the character from proceeding forward with the same velocity, hence it is possible to determine the position in the floor directions, by applying rules of a projectile. If the last know velocity is denoted v,the acceleration is denoted a and the angle to the floor plane we denote θ. We find the new position on the floor-plane p as

$$p = \mathbf{v} \cdot t \cdot cos\theta,$$

and the vertical position as

$$z = v \cdot t \cdot sin\theta - \frac{1}{2}gt$$

where $a = (0, 0, g)$ and t is the time from when a foot last touched the ground.

9 Alignment Curve

When using linear blending after finding the timewarp curve, we still suffer from one of the problems with the linear blend. Given a blending between two motions with a rotation, the linear blend fails once the angle between the motions are larger than 180 degrees.
An example of this problem can be seen when blending two equal walking motions which are turned in two different direction. Since they are two equal motions, and they rotate at the same angle about the vertical axis, we expect a walking motion in a straight line.
This problem is illustrated in figure 15, where it is seen that by not considering these differences in orientation, the linear blending for the timewarp curve will result in the character flipping around once the accumulated angle between the two motions exceeds 180 degrees. This is due to the fact that the linear blend chooses the smallest angle, and therefore when dividing figure 15 into large steps, we get a forward motion as seen in figure 16 step a). When we hit a step which brings us above 180 degrees (step b)), the orientation changes (step c)).

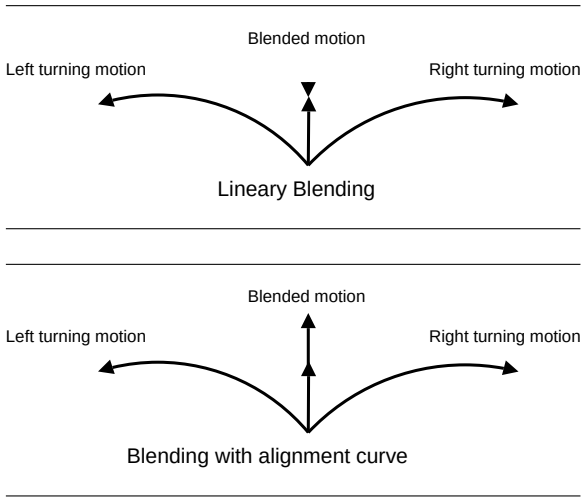

Figure 15: Blending of two motions turning away from each other. The arrows mark the direction of the character. With linear blending the character turns around instead of proceeding forward. The lower picture shows the blending after adding the alignment curve

A solution to this is the use of alignment curves. By using this we can achieve the situation from figure 15 where the blended motion does not turn around.
As discussed in section 7 motions are fundamentally unchanged about the vertical axis (the z-axis).

The alignment is made by letting each motion have a vote on the position and the rotation and these votes put together with the blend weights will decide the final position and orientation.
This should only be applied to the root, since all the other bones change according to this.

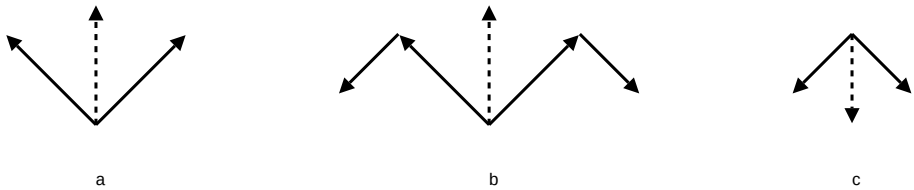

Figure 16: Step a) shows the motions before they hit 180 degrees and how the orientation is forward. Step b) shows the step above 180 degrees and step c) shows why the orientation changes

In [7] an alignment curve $\mathbf{A}(u)$ are created, that gives a set of transformations for each point $\mathbf{S}(u)$ on the timewarp curve. Since we dont use a curve, we change the memory containing to fulfill the alignment constraints.
When calculating the timewarp curve, we found the angle θ that could be used for transforming the destination frames local coordinate system onto the source frames coordinate system. This value can be used here by using this as the weighted orientation for deciding the final orientation. In order to make sure no angle is above 180 degrees, we use the alignment constrain that $|\theta_i - \theta_{i-1} \leq \pi|$. To fulfill this we change θ_i for the i'th frame on the timewarp curve if this criteria does not hold. When an angle passes 180 degrees θ_i changes sign, hence we continue counting in the same sign as θ_{i-1}, but adding the difference of θ_{i-1} and $-\theta_i$.

10 Implementation

As mentioned in the introduction, section 1, this project should work as part of the OpenTissue project. This means that some of the theory mentioned in this report have already been dealt with in OpenTissue and have been implemented as part of this project, and as such we use these components directly for implementing our registration curves. This includes a skeleton structure for building characters and a linear blender. The task is to replace this linear blender with the implementation of registration curves.

For doing so three classes has been constructed. Two classes should calculate the blends between all available motions, and hence be calculated during the initialization and one class should be the one evaluating the information from the path and hence creating the final blend. The precomputed classes consists of a `sequence` class that holds information about the sequences used in the blend, and a `blender` class that performs the calculations and constructs the registration curves that can be extracted from the `graph blend schedular` class which controls the blending process.

All source code is included on the cd enclosed with this paper. They are found in the folder `src/`. Comments show which code parts or classes I take responsibility for.

10.1 Sequence class

The sequence class holds informations about a sequence and can be seen in the file `sequence.h`. This class is saving original information about the animation, so the it is later easily recovered. The sequence is divided into a number of frames by using a user determined step size, and for each of these frames, the class saves information about position, rotations and weights. The information can be extracted from the `sequence` class by either using a frame number or a time stamp. If a time stamp has been given, the sequence function will translate this into the best matching frame number and lookup from this. The other way around it is also possible to get a certain timestamp by giving a frame number.
Since the motion has to run through all of its time steps in order to save information, it is natural if we save the mean value ($\bar{p}$, equation (2)) being used in the timewarping algorithm (section 7). It should also contain information about whether the motion is cyclic or noncyclic.

10.2 Blender class

This class creates the registration curve, which includes calculating the distance map, finding path and aligning this path. The distance map is represented by another small class (`array2D.h`) whose only task is to hold a two-dimensional array of given information. The blender class can be found in `blender.h`.
When the registration curve has been computed, the `blender` class holds a copy of the distance map and two paths, one for each motion in a blend, so we for example don't have to calculate a distance map for both walk to run and run to walk. We simply flip the distance map.
The blender also creates the alignment curve, by editing the θ values as described in section 9. The registration curves are computed when starting the program, and can have a long run time while calculating. But once loaded the blending can be performed by simply making a

cheap lookup in the path.
We handle both transitions an blending. We use blending when we change into a noncyclic motion. Then we wish to blend while the non-cyclic motion computes and then return to the original motion, because a non-cyclic motion only should be computed once. This is ensured by the fact that our path covers the whole of the destination motion and thus we can cross fade the blending so the destination motion blends in with a weight of zero, peaks at the weight one halfway through the path and counts down to zero again.
If both motions are cyclic, we perform a transition with cross-fading blends from zero to one and ending with only the destination motion.

10.3 Graph Blend Schedular class

The schedular class constructs the animation of the motions and can be found in `graph_blend_schedular.h`. When only one motion is active it runs this by using a global time and finding the position of the character for this specific time. When two animations are active, it makes a lookup in the registration curve on whether this particular local time of the animation has any match on the path. If not it gets the closest point of when to begin the animation, and continues only performing the first animation until it hits this point. On being blended into a non-cyclic motion, the blender class tells the schedular class to wait until it hits a point that matches the beginning of the destination frame.

When having found a point on the path, the schedular class extracts the index numbers to the frames, sets the weights according to the length of a transition and performs a linear blending using the frames extracted from the sequence class using the indexes from the path. The rotation is changed to be according to the angle θ, also data given from the path.

The program only handles two motions, so when more is added we perform a regular linear blend on all of them.

10.4 Other files

Small changes has been made to other files already in the OpenTissue project.

We have made it possible to rotate an animation with focus on testing alignment curves. This feature appears in two functions. One is where we read the XML-files containing the motion information, seen in the file `keyframe_animation_xml_read.h`.
Here we add an argument of a given angle to rotate the constructed motion. If we want to extend this animation further a function has been added to `keyframe_animation.h`, where we give a rotation angle and how many times we want to repeat the motion. This is only needed for rotation, because other motions are cyclic and thus do not have to be repeated.

11 User guide

To construct a motion blend, a demo is constructed under OpenTissue, in `demos/opengl/MotionGraph/`.
In the file `application.cc` and `application.h` it is possible to load the motions from which we would like to perform an animation and to define the time step both for which the animation should run and for which the motion graph should be computed. All motions has to be followed with information about their weight according to the other motions and whether they are cyclic or non-cyclic.

We might wish it possible to change the direction of one of our motions, thus making it turn instead of only walking straight. This can be done when initializing the motion. A motion is initialized reading data from a XML file
`keyframe_animation_xml_read(filename,keyframe_animation,angle)`,
where `filename` is the name and position of the xml-file, `keyframe_animation` is the address where we save the motion and `angle` is the turning angle applied to the motion.

To initialize and load all motions and create the registration curve, we press 0 once the program has started.
The motions are assigned to the keyboard strokes 1 to 9. When having initialized the motions, a stroke on one of these keys will either turn the motion off or on in the animation. When more than one motion has been turned on, a motion transition is started. For now the code is only set to perform transitions for two motions.

12 Testing

In this section, the developed code of the registration curve will be tested. Testing will show on whether the registration curves solves the problems we try to handle, and if each part of the registration curve works as desired, and thus testing whether we have created a generic approach for handling blending of motions.
Some of the testings are done by the use of filmstrips, which can all be found on the CD enclosed with this paper in the folder `movies`. All filmstrips are avi-files compressed with DIV-X codex.

We test registration curves on combinations of 4 different motions. These can be seen in the folder `movies/original_motions/`, as the files `still.avi`, `run.avi`, `strut.avi` and `walk.avi`. Note that the strut-file has a small error in the motion so the last point of the motion does no match the start exactly. This makes it a non-cyclic motion, but we still use it as a cyclic. The error lies in the motion used, and thus has nothing to do with the registration curves.
Further more we will se testing on partial motions, where only a subset of the skeleton is defined. But these are not included as a video file showing the animation before blending, since they only look correct when blended with a cyclic motion.

12.1 Distance maps

The timewarping algorithm is divided into two parts that needs testing. One is the creation of a distance map, that holds the distances between each frame of the blended animations, and the other is the path and how to use it. This section tests the distance maps.

In testing the distance map, we look at some constructed maps to see if they seem to be correctly constructed, hence having high and low values where we expect it.

Equal motions
A simple but crucial test for the construction of a distance map, is to create a map where the destination and source motions are the same. Then the diagonal of the distance map should be zero.
The test is conducted by using two equal running motions. The gray scale map is printed, where the lower the distance are, the darker the color on the map will be. In figure 17 we see the distance map is as we expect it to be, since it has the diagonal as the lowest values. In the right picture we brighten up the picture, so it is easier to see where the lowest values can be found.

We test the theory on whether the path is as we expect it to be by inserting the computed path into the distance map. In figure 18 we see the same distance map as is figure 17, but with a correctly computed path along the diagonal, represented as white pixels.

Non-equal motions
The challenge comes when blending two non equal motions. We try making a map of a blending between a running and a strut-walking motion. The result is seen in figure 19, where we again add a brighter version of the map to better show the details.
The smallest distance found in the map is in the point (100,27), and the position of the two characters look as in figure 20, where the red character is found at the destination frame and

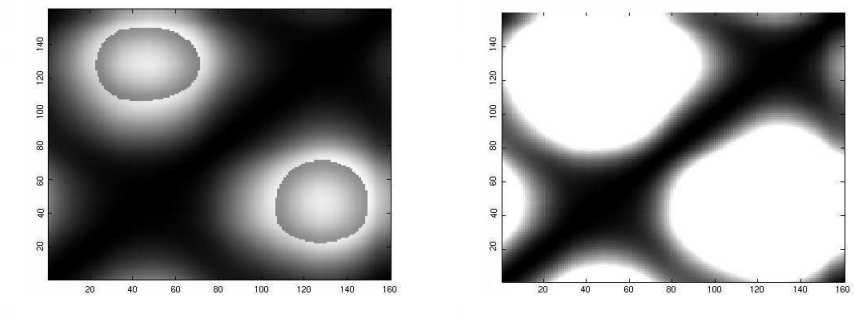

Figure 17: The distance map of two equal motions. The darker the color, the lower the distance in that particular frame. We see the correct dark diagonal which implicates a distance of zero. To make this clearer, we brighten the result in the left picture.The map is made with a time step of 0.005

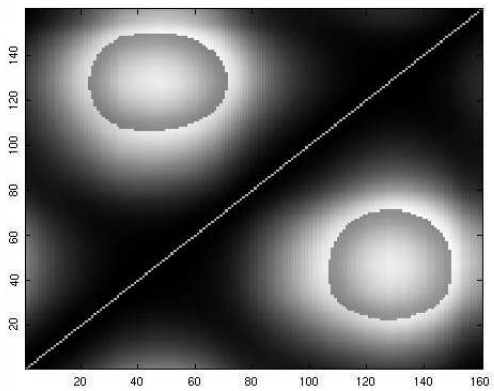

Figure 18: The computed path along the diagonal of a map computed from two equal motions. The path is represented by white pixels.

the grey at the source frame. It is easy to see that the characters are much alike.
Equally we see the worst fit in point (53,104) in the distance map, and as seen in figure 21 it is believable to be the worst fit, due to the fact that the grey frame has the right leg in front and the red has the left in front.

The blending of a strut-walking motion and the running motion as seen above is a blend between two quite similar motions. We now try blending two not so similar motions in the form of a run blended with a still standing motion. It is expected to clearly be only two times the animation matches, and this is when the legs are centered. See figure 22.
Both the worst and best fit will result in an equal still standing motion as seen in figure 23. The best fit of the running motion is as expected with the legs close together and the worst

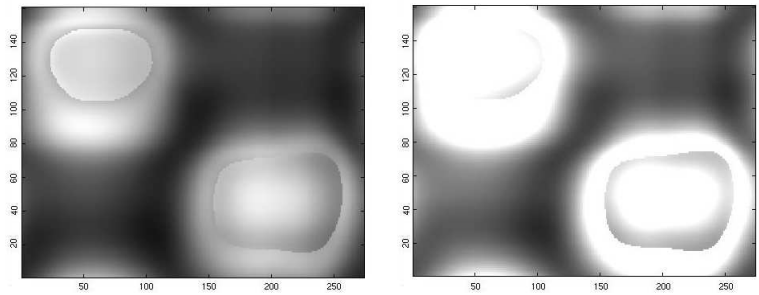

Figure 19: The distance map of a running and a strut-walking motion. On the right we see a brightened version to bring out the details. The map is made with a time step of 0.005

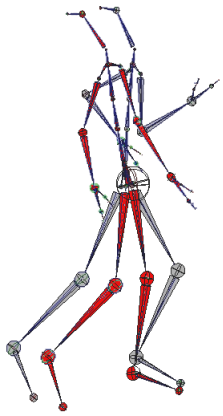

Figure 20: The smallest distance of the distance map in figure 19 is found to be these two character positions. The grey character is from the running motion and the red character is from the strut-walking motion.

with the legs far apart. See figure 24, where the best fit is seen on the left and the worst on the right.

Even though it seems like the map functions correctly, we test the transformation used to make the transformation of the destination motion onto the source motion.
We use a still standing motion and rotate it to see if the calculated angle fits according to the rotated. Here both rotation and translation parameters were as expected.

12.2 Timewarping path vs. linear blending

When having tested that the computed distance map to be believable, we test the paths found from this map. In doing so we add comparison to an ordinary linear blending, in order to see how the the path creating a timewarping improves the animation. To create a better

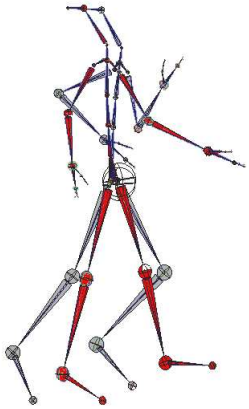

Figure 21: The largest distance of the distance map in figure 19 is found to be these two character positions. The grey character is from the running motion and the red character is from the strut-walking motion.

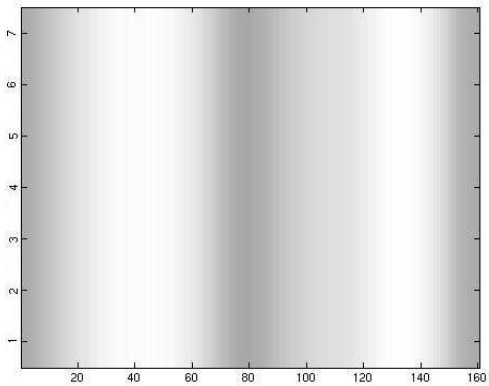

Figure 22: A blend of a still standing motion into a running motion, where the running motion is seen at the x-axis and the still motion at the y-axis.

overview of the details, we perform the animations without root displacement.

Blending running with strut-walking

We perform a linear blending without using using the registration curves developed in this project. We start by examining a blend between a running motion and a strut-walking. All movies concerning blending between running and strut-walking can be found in the folder `movies/run_strut` In the file `run_strut_linear.avi` we see a linear blending of these two motions. There are two problems in this animation. These problems are shown in the files `run_strut_linear_shift.avi` and `run_strut_linear_timing.avi`, which are segments of

Figure 23: Still standing motion has equal frames on both the best and worst fit.

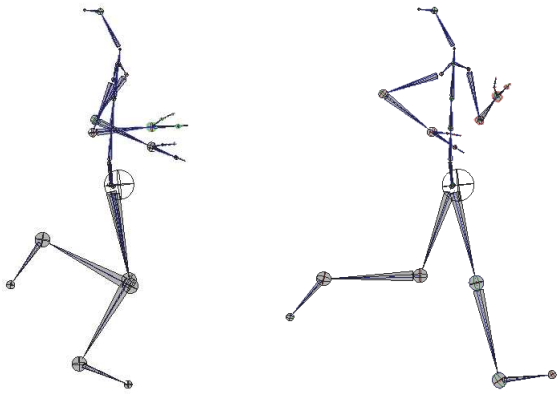

Figure 24: The best and worst fit of the running motion when blending with a still standing motion. The best fit is on the left and the worst is on the right

the total filmstrip of the linear blending.
Looking at the filmstrip `run_strut_linear_shift.avi`, shows the problem that occurs when no best starting point is found for beginning the blend. The linear blender starts the blending instantly and uses the first frame of the new motion to blend with.
The other problem becomes apparent in the filmstrip `run_strut_linear_timing.avi`, where we se how no timing results in one motion moving remarkably faster than the other, hence giving the same problem as the start where the motions being blended are in remarkably different positions.
Both problems are being solved by the use of timewarping. By performing a timewarping with a slope limit of 1, it is possible to se a solution to the problem concerning a reasonable

starting point but still having the timing problem. The path can be seen in figure 25, where we see how the path are forced to enter areas with a bad distance.
We should now be able to see the problems when we are passing through an area with high

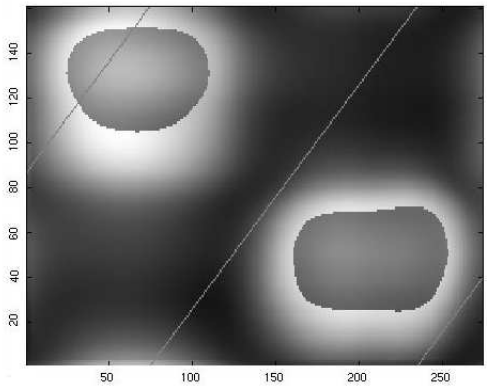

Figure 25: Timewarping with slope limit 1 used to show linear blending with a reliable starting point

distance between the motions. When modifying the blend weights as described in section 4.2, this problem is not visible, since the large distances are so close to the start of a blend. This mean that the source animation has such a heigh weight that the destination will not yet have any influence (see filmstrip `run_strut_time1.avi`). Hence we remove this blend weights and thus having a direct blending where both motions have same weight from the start of the blend. This can be seen in `run_strut_time1_noblend.avi`. We should remember that these animations are not completely alike, but the result is still at lot better than in the linear blending. It is here clear that the animations are in the same position. We also see that in the beginning we have some timing problems, just as expected, but after this the timing look reasonable, also as expected from the path in figure 25.
To resolve the timing issue we use a full timewarping, where we add a slope limit larger than 0. As seen in figure 26 the path with a slope limit of two is already remarkably better, and as soon as we use a slope limit of three, the path looks good and it should not be necessary to try higher slope limits. We must consider the fact that the higher a slope limit the slower the animation will be and this could look unnatural if the animation slows remarkably down once the blend is started.

We examine the filmstrips of the two animations with different slope limits and with/without blend weights.

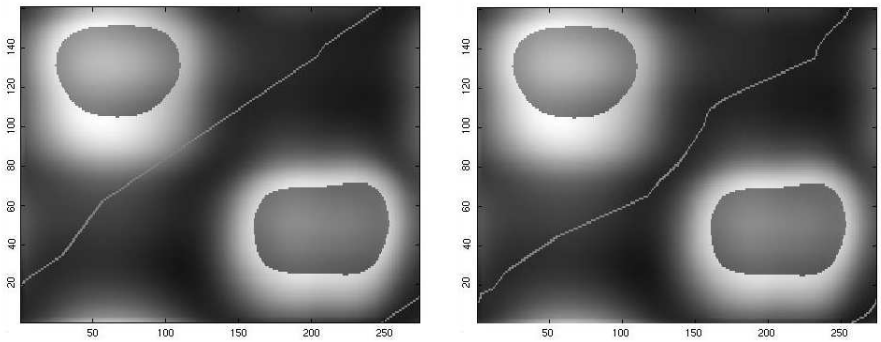

Figure 26: Two equal distance maps with the paths found by using different slope limits. In the left picture we see a path using slope limit 2. We see that the path is not yet perfect, where it with a slope limit of 3 (right picture) are far more as we expect it to be.

Slope Limit	**No blend weights**	**File**	**With blend weights**	**File**
2	By running with no blend weights, we expect to see the point where the path runs through a white area. This is not the case though. Both the start and the animation hereafter looks very good considering no weights.	run_ strut_ time2_ noblend.avi	The transition with blend weights look as realistic as expected from the test with no blend weights. This slope limit delivers a good result.	run_ strut_ time2.avi
3	We see an almost perfect blend even though we have removed the blend weights. We can still clearly see where the blending was initiated, but the starting point is as close to perfect as comes. The blend after the starting point bears no sign of lack of timing.	run_ strut_ time3_ noblend.avi	As in the film of slope limit 2 the walk with blend weights added looks very reliable, which was to be expected considering the path as seen in the right picture of figure 26. A problem with this slope limit occurs because it slows down the animation in such an extend that it looks odd.	run_ strut_ time3.avi

As discussed in the tabular a slope limit of two for creating the path gives a very reliable result, and it is shown how the use of timing and a good starting point improves the reliability of the animation.

Blending running with walking
In blending running and walking, we blend two animations that look more similar than strut-walking and running. Due to this fact we expect better results while using linear blending and an almost perfect result when using timewarping with slope limit 1 or 2. In figure 27 we consider the paths of the slopes 1 and 2.

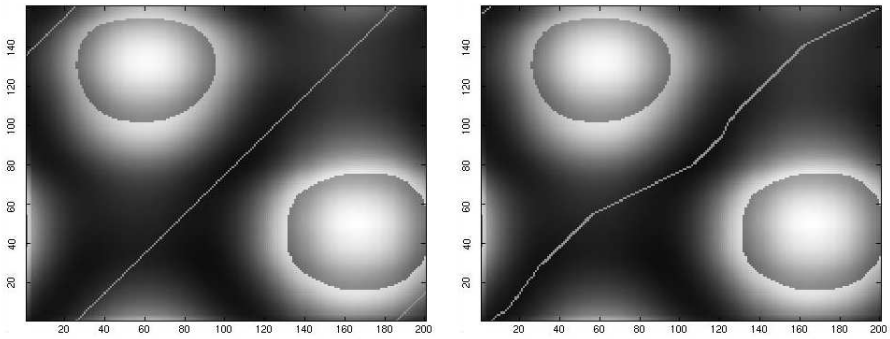

Figure 27: Two equal distance maps for the blending of a running and walking motion, with the paths found by using different slope limits. In the left picture we see a path using slope limit 1. We see that the path is not yet perfect, but since the motions are alike, it is close. On the right we se a path with slope limit 2, and here the path looks to be as good as it gets.

We now evaluate the blending between running and walking for a linear blending, and the path with slope limits 1 and 2 for both with and without the use of blend weights. All film clips used in this section can be found in the folder `movies/run_walk`. Even though the animations are much alike, we still see some problems in the linear blending. The problem still arises that when the blending are started with the animations at different positions it gives a bad start blend. Because these motions are alike, a linear blend continues being off when animation continues. This can be seen in `run_walk_linear.avi`.

We now consider the timewarping

Slope Limit	No blend weights	File	With blend weights	File
1	As seen in the left picture in figure 27, the optimal path is already rather good. This should result in a good starting point and a good blend after this.	run_walk_time1_noblend.avi	The good result should give a very believable transition	run_walk_time1.avi
2	The path with a slope limit of two is a bit better than the one with slope limit of one (as seen in figure 27, but it should not be so much as to be visible in the animation.	run_walk_time2_noblend.avi	We expect an animation that reminds us of the one found above for a slope limit of one, since the paths are alike.	run_walk_time2.avi

From working with the timewarping, experience shows that a slope limit of two holds for all cases (also some not mentioned in this testing), hence setting this permanently to two will keep the generic approach as was the purpose of this method.

12.3 Changing time step

Until now we have proved that the timewarping algorithm works when we are computing the distance map with the same time step as the one being used when computing the animation. It is also possible to animate with a different time for the animation and the time used to determine the distance map.

If the time becomes larger, we expect the animation to be as good as when running the animation with the time step used for computing the map. This is due to the fact that we do not have to ease on the quality. We probably have to skip some frames. Since we do not use a curve approximation, this could result in some jumps in the animation if the frames are not skipped evenly. An animation has been created where the distance map has been computed with the time step 0.005 and the animation has been computed with the time step 0.001. The result is seen in `run_strut_005_001.avi` in the folder `movies/different_stepsize`. This example is very successful.

The problem becomes larger once the time step is smaller than the one used to compute the map. Then we expect to see some frames repeated, which should result in a non-smooth motion. This can be seen in `run_strut_005_001.avi`, where we use a time step of 0.005 to create the distance map and a time step of 0.001 in the animation.

We can conclude that if the animation time is adjusted according to the time used on the distance map, the two times should not be far from each other when the animation time is smaller than the blend calculated time. The other way it is okay to have a big time difference. If using a bigger time difference, an approximation of a curve like in [7] would be optimal. A way of still keeping this structure where a simple lookup is enough and though improving, would be to create linear line segments between the path segments. This should be easy to add to the current structure.

12.4 Alignment

Until now we have only tested timewarping. The alignment is added when we pass 180 degrees between the two characters being blended. Thus we create a test where we make a rotation motion and blend it with the same motion, rotation in the opposite direction. To make the case as simple as possible we use a still standing motion. These can be seen in the folder `movies/original_motion/` as `rot1.avi` and `rot2.avi`.
Before adding alignment, we see the blended motion turn around. See `noalign.avi` in the folder `movies/alignment/`. When adding the alignment we see the blended motion is keeping the correct direction through the whole animation. This can be seen in `align.avi` in the same folder.

12.5 Root motion

The criteria for adding root motion was that the feet should not be sliding on the floor. This should be tested for both the original motions and the blended motions.

12.5.1 Original motions

Starting with the original motions, we test movement for a running, a walking and a rotated walking motion. We note that the feet are static on the floor, meaning a correct root movement. See the filmstrips `run_move.avi`, `strut_move.avi` and `strut_rot_move.avi` in the folder `movies/movement/`.
We do however see some foot-skating in `walk_move.avi`. This is due to the fact that both feet are in the ground at it seems as if we are not choosing the correct one to hold steady in spite of trying to determine which should be chosen from the direction of the feet.

12.5.2 Blended motions

Some problems occur when blending and moving the root. When finding the best match between two characters, we have no guarantee that the best match is not in the position behind the current position. This could result in a short backward going motion at the start of the blend, which does not go well with moving the joint. In `run_strut_move.avi`, we se this as the animations starts, and how a slow initial velocity at setting off the free-fall of a run, results in an almost static movement in the air. But other than the short moment where the blending starts, the movement works for blending as well. In `still_run_move.avi` we also see a bit of starting problems due to the same issues as mentioned above. The two motions have very little in common and thus it is hard to start the moving of this blended motion.
A good result however is the `run_walk_move.avi` because the motions are so much alike that the start of the blend will not be remarkably different than the source animation.

12.6 Blending with partial motions.

Given a partial motion, where only a subset of the nodes are defined, it is still possible to perform a blending. We test a wave and an arrow-shooting motion. They are both non-cyclic, and as mentioned in section 3, we only wish to perform such a motion once. We therefore expect an animation starting with the cyclic motion, and then using cross-fading blending where the new partial motion starts with a blend of zero, and at the middle of them motion it reaches its maximum blend, from where the weight counts down to zero to end with only having the cyclic starting motion left. This can be seen in the folder `movies/partial/`, in the files `walk_wave.avi` and `shoot_arrow_move.avi`, where both a moving and a static animation is shown.

13 Conclusion

A successful registration curve including timewarping and alignment has been computed.
It is possible to us one time step for computing the distance map and another for running the animation. This is however not perfect. It would benefit from being approximated to a curve or at least a linear function.
We have shown generic blending of different animations, both cyclic, non-cyclic and partial. All done without changing settings. We do however have to inform the program on whether a motion is cyclic or non-cyclic.
A correct movement has been constructed, though with some problems when blending, but this is due to correct choosing from the blending that does not always ensure that the best fit will be found as a starting point of a position further ahead in time. This could be interesting to experiment with.

An alignment has been constructed, which ensures that the animation does not flip around once the two blended motions passes 180 degrees.

The registration curve has been compared to an ordinary linear blender, and shows remarkable improvement of this, so the project is overall considered a success.

A further addition would be to handle blending of more than two motions, which I didn't have time to implement in this project.

I was of the believe that a distance map should have smooth curves, so I was suprised when they didn't (see for example figure 27). Though if we look at the map computed in figure 22, we see a map with no sharp corners. I can only assume that I must have misunderstood that smoothness was required, since I have conducted several succeding tests, both of the parameters found to calculate the map and on animating the final animation.

References

[1] Armin Bruderlin and Lance Williams.
Motion Signal Processing
In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 97-104. ACM SIGGRAPH, August 1995.

[2] Anthony C. Fang and Nancy S. Pollard.
Efficient Synthesis of Physically Valid Human Motion
ACM Transactions on Graphics, 22, 2003.

[3] James Foley,Andries van Dam, Steven Feiner and John Hughes.
Computer Graphics. Principles and Practice
1990 by Addisom-Wesley Publishing Compagny, Inc.
ISBN 0-321-21056-5

[4] Michael Gleicher, Hyun Joon Shin, Lucas Kovar and Andrew Jepsen.
Snap-Together Motion: Assemblingrun-Time Animations
In Proceedings of the 2003, Symposium on Interactive 3D Graphics, April 2003.

[5] Rachel Heck, Lucas Kovar and Michael Gleicher.
Splicing Upper-Body Actions with Locomotion
Computer Graphics Forum (Proceedings Eurographics) - winner of 2nd prize best paper award. September 2006.

[6] Lucas Kovar and Michael Gleicher.
Automated Extraction and Parameterization of Motions in Large Data Sets
In *Proceedings SIGGRAPH 2004*. August 2004.

[7] Lucas Kovar and Michael Gleicher.
Flexible Automatic Motion Blending with Registration Curves
In *Proceedings of ACM SIGGRAPH /Eurographics Symposium on Computer Animation 2003*. July 2003.

[8] Lucas Kovar, Michael Gleicher and Frédéric Pighin.
Motion Graphs
In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series, pages 97-104, ACM SIGGRAPH, July 2002.

[9] Jehee Lee, Jinziang Chai, Paul A. A. Reitsma, Jessica k. Hodgins and Nancy S. Pollard.
Interactive Control of Avatars Animated with Human motion Data
In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series, pages 97-104, ACM SIGGRAPH, July 2002.

[10] The OpenTissue project.
a low level application programming interface (API) and a playground for future technologies in middleware physics for games and surgical simulation.
www.opentissue.org

[11] Ken Perlin.
Real Time Responsive Animation with Personality
IEEE Transactions on Visualization and Computer Graphics, 1(1):5-15, March 1995.

[12] Arno Schödl, Richard Szeliski, David H. Salesin and Irfan Essa.
Video Textures
In *Proceedings of ACM SIGGRAPH 2000*, Annual Conference Series, pages 489-498.

[13] Ken Shoemake.
Animating rotation with quaternion curves
In *Proceedings of ACM SIGGRAPH 85*, Annual Conference Series, pages 245-254. ACM Press, 1985.

[14] Jing Wang and Boddy Bodenheimer. 2004
Computing the Duration of Motion Transitions: An Empirical Approach
In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2004.

[15] Andrew Witkin and Zoran Popović.
Motion Warping
In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 105-108. ACM SIGGRAPH, August 1995.