Synthesis of motion capture data

Tomáš Sako

2008

Comenius University
Faculty of Mathematics, Physics and Informatics
Institute of Informatics

Tomáš Sako

The synthesis of the motion capture data

Diploma thesis
Thesis advisor: RNDr. Stanislav Stanek

Bratislava 2008

I honestly declare, that I disposed the diploma thesis on my own, with the use of the literature and resources mentioned in the end of the thesis.

Bratislava, May 7th 2008 Tomáš Sako

The goal of this work

Our goal is to examine the methods of motion capture data synthesis (editing and manipulating this data). The result of this work should be a program, which takes motion capture data as input and the output should be their composition created by semi-interactive manner, so that the resulting motion is as realistic as possible.

Acknowledgements

I would like to thank to my thesis supervisor RNDr. Stanislav Stanek for many helpful advises and ideas. A huge gratitude belongs also to my friend Miroslava Božeková for her detailed reviews and to my family for their patient support.

Abstract

Title: The synthesis of the motion capture data
Author: Tomáš Sako
Department: Department of Applied Informatics
Advisor: RNDr. Stanislav Stanek

Abstract: The introduction of this work deals with the most widely used methods of motion capture data synthesis. Next, we present method called 'Timewarp curve' derived from 'Registration curves' proposed by Lucas Kovar and Michael Gleicher and make our modification of this method, which should speed up motion data synthesis. We implement mentioned method and compare it with another in our software product (Motion Blender). As the result, we get the comparison of efficiency, speed and reliability of those algorithms. We change original 'Timewarp curve' and get a novel algorithm that preserves realism of human motion. In the end, we describe the structure, features and usability of the program and show the results of experiments.

Keywords: motion capture, motion blending, registration curves, motion synthesis

Contents

List of Figures

List of Tables

1 Overview of the chapters

Chapter 2 gives the outline of the problem, discusses aims of this work and describes what have been done yet.

Chapter 3 contains the definition of solved problem

Chapter 4 contains complex description of method - Registration curves, by which we were inspired.

Chapter 5 consists of our benefit - the summary of methods, which we have proposed.

Chapter 6 deals with software implementation of all methods : program structure, features, interface, etc.

Chapter 7 gives a complete overview of our experiments and there is also shown the correctness of our proposed methods.

Chapter 8 concludes this thesis and discusses possible future directions.

2 Introduction

2.1 Definition

Motion capture (Mocap) is a technique of digitally recording movements for entertainment, sports and medical applications. It consists of tracking points on a moving face or body in order to get a simplified image of motion[1].

2.2 Motivation

It is very difficult to animate human motion. The reason is that not only the motion itself is complicated but also our familiarity with this kind of motion is very huge. That means that we are very sensitive to even small artifacts or errors in human motion, because we see moving people each day for many times. Motion capture is the technology that enables us to get an arbitrary data of realistic human motion performed by the actor. Unfortunately, we often need to make some changes to the captured data in order to get exactly what we want. Studios equipped with motion capture technology are very expensive and therefore we want to be able to edit our data without the loss of fidelity.

In recent years, there have been many approaches how to manipulate captured data in order to facilitate and speed up the work of animators. Many problems have been solved, not only editing of data, but also realistic joining motions together and parametrizing of motions (i.e. we can control the strength and target of the punch or set trajectory of walking person). This thesis deals with the second mentioned problem, that means realistic combining and mixing of captured human motions in order to get satisfying result. It is called motion blending(i.e. we have two motions of running and walking person and we would like to make a jogging motion so that its pace is higher than the pace of walking but lower than running motion).

Our goal is to examine the methods of motion capture data synthesis (editing and manipulating this data). The result of this work should be a program, which takes motion capture data as input and the output should be their composition created by semi-interactive manner, so that the resulting motion is as realistic as possible.

[1]en.wikipedia.org/wiki/Motion capture

2.3 Our benefit

We present a method called 'Timewarp curve' derived from 'Registration curves' proposed by Lucas Kovar and Michael Gleicher and make our modifications of this method, which should speed up motion data synthesis. We implement the original method and compare it with our proposed modifications in our software product (Motion Blender). As the result, we get the comparison of eficiency, speed and reliability of those algorithms.

The benefits of this work are : creation of the overview of motion synthesis methods, creation of a semi-interactive system for motion blending and finally our proposed methods with experiments. These methods are attempts for the improvement of existing methods of timewarping(correct timing of the resulting blended motion). We experiment with a timewarp curve, edit and smooth it and compare dicreet and continuous(B-spline) approaches.

2.4 Previous works

In the past, many works were focused on realistic motion blending. First approaches were based on keyframing, because it is the most simple and therefore do not need many computations. Recent approaches were mostly based on physics, because better hardware provides higher computational power, which is essential for these approaches.
However, there is still no universal method for getting such a result from motion blending, that would be unrecognizeable from the real movement of person. Therefore we propose a few modifications that should help to approximate to an ideal.

Standard classification of motion synthesis techniques is : Methods Involving Physics, Hand Controlled Methods, Data Driven Methods.

2.4.1 Methods Involving Physics

As the physical laws influence the motion of humans, there are several approaches which implement such laws into motion synthesis. What we need is mass distribution for the entire body, the joint torques and knowledge of Newton's laws. We can find average mass distribution in biomechanics [7]. Problem with torques solved Hodgins et al. [8] by using finite state machines and proportional-derivative servos. Faloutsos et al. [9] attended

to motions for preserving balance. Disadvantage of this approach is that controller design is difficult to perform and such controller can produce only severe motions. On the other hand, when it is generated, we can change the input circumstances and produce particular motion like Laszlo et al. [10] did. Hodgins and Pollard [11] adapted a controller to a new body by computing controller parameters with scaling and consecutive tuning the results using simulated annealing. Another method uses a few keyframes and adapts physical laws to motion resulting from simple interpolation. Liu and Popovic [12] worked with ballistic motions and used spline interpolation. Character situated in the air obeyed Newton's laws and on the ground model of momentum transfer. Fang and Pollard [13] did it more effective, when they have shown that physical constraints in the form of aggregate force and torque can be differentiated in time linear in the number of DOFs.
Sometimes, physical approach generate motions with lack of personality. Neff and Fiume [14] implemented the fact that opposing muscle forces varied the amount of tension. However, there is still no physically simulated method that would provide an arbitrary motion that would be realistic.

2.4.2 Hand Controlled Methods

This is the oldest technique, where the animator specifies individual degrees of freedom (DOFs) and joint torques at some points in time, which are called keyframes. Other data in between keyframes are computed by simple interpolation methods. The main disadvantage is that animator has to create the frames manually, which is very tedious work. On the other hand he has the full control over motion. The more details the animator designs, the more convincing motion he gets. Many poses of the character are needed, while 24 frames per second is considered as optimal frame rate. There is also another technique which uses algorithms, that procedurally replicate motions. It is the way, how to manually create motions at once. Perlin [3] and Perlin and Goldberg [6] have shown that many motions could be generated with simple and efficient algorithms. Disadvantage is that the most of edited motions have lost the realism.

2.4.3 Data Driven Methods

Invention of motion capture technology has brought realistic example motions of high fidelity, which are used in data-driven synthesis algorithms. However, example motions can be generated by keyframing or physical simulation. One possibility is signal processing operations apllied to each DOF. Bruderlin and Williams [15] introduced some operations like multiresolution

filtering, waveshaping and adding smooth displacement maps. Witkin and Popovic [16] proposed motion warping and Gleicher [17] used displacement mapping to have an interactive control over character's trajectory. Problem of these algorithms is that they fail when more body parts must be adjusted simultaneously.
There were also some approaches to have a full control of the motion's style and aesthetics. Unuma et al. [18] worked with cyclic motions and linearly combined the Fourier coefficients of DOFs and found out that it is possible to control the emotions in human movement. Tak et al. [19] checked the position of the body's zero moment point to preserve physical validity. Popovic and Witkin [20] built physically-based framework for editing, that mapped original motion onto a simplier model.
In this category belong also motion blending, motion graphs and parameterizing motions, which will be closely characterized in the next section.

3 Basics and definition of the problem

Our figure in motion is represented as the skeleton consisting of *joints* and *bones* which connect two joints. Bones are rigid and they can not bend or change their length as in the real world. Joints represent the flexible parts of human body. We keep joints in the hierarchy of tree, because it is easily represented in computers. Each joint has one parent (except the main joint) and joint's position is given relatively to coordinates of its parent. This is called the *offset*. Main joint which does not have parent is called *root*, it mostly represents the pelvis or spine and its position is defined relatively to the world coordinate system. All body parts are represented at the picture below (see figure 1).

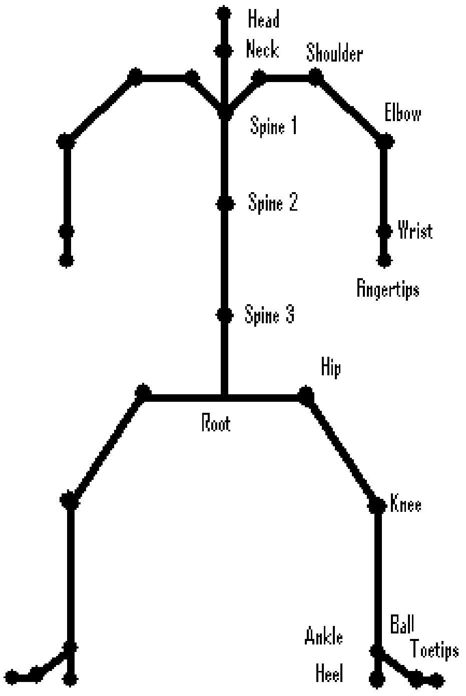


Figure 1: The picture of skeleton

Each motion is represented as the function of time, that returns frame specified by the position of the root and rotations of individual joints :

$$\mathrm{M(t)} = \Big(\ \mathrm{p(t)},\ q_1(\mathrm{t}),\ .\ .\ .\ ,\ q_n(t)\Big), \tag{1}$$

where p is the position of root in world coordinate system and q_i is the orien-

tation of the i-th joint relative to its parent's coordinate system. These orientations are mostly represented by unit *quaternions* or Euler angles (which we use).
As input we have data from mocap - set of frames, where number of frames depends on frame rate. In motion are frames represented by skeleton configurations (poses) M(t_1),..,M(t_j) corresponding to regular sampling of our motion M(t). Unfortunately, we do not have all frames needed to build particular motion. Frames, that are measured from mocap, are then interpolated in order to generate in-between frames that are missing. Animators mostly use linear interpolation to generate root positions and spherical linear interpolation (slerp) to generate intermediate joint orientations.

Helpfull tools in motion blending process are parameter curves. They represent the values of rotations or positions of joints in time progress. We can investigate properties of motion(continuity, sudden changes in movement, satisfying constraints) with them. See figure 2

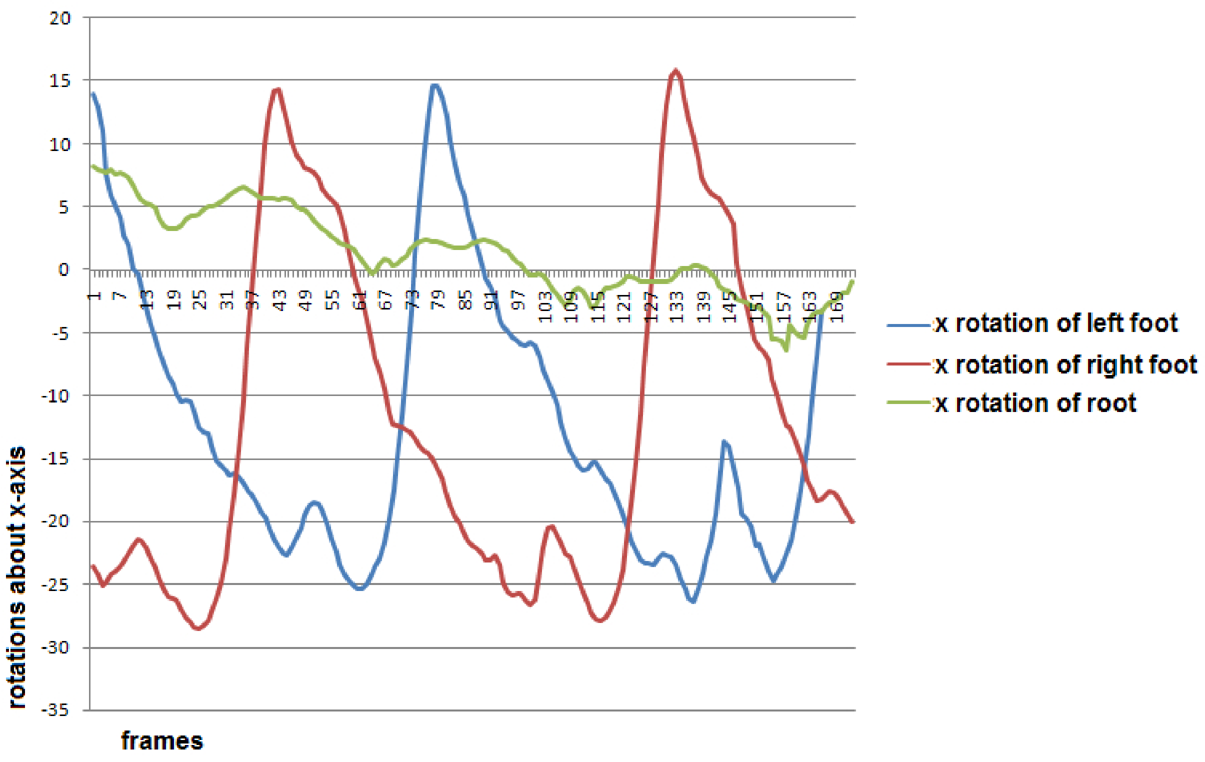


Figure 2: Example of parameter curves with angles (in degrees), extracted from running motion

3.1 Motion blending

Let us have two input motions defined as functions $F(t), G(t)$. As we would like to interpolate between them and find some meaningful in-between frames, we can use simple equation :

M(t) $= \alpha F(t) + (1 - \alpha)G(t)$ (2)

This is called weighted average of the poses, where weights are determined by α value. However, this simple approach often does not bring plausible results, for example see figure 3. Because of these artefacts, another algo-

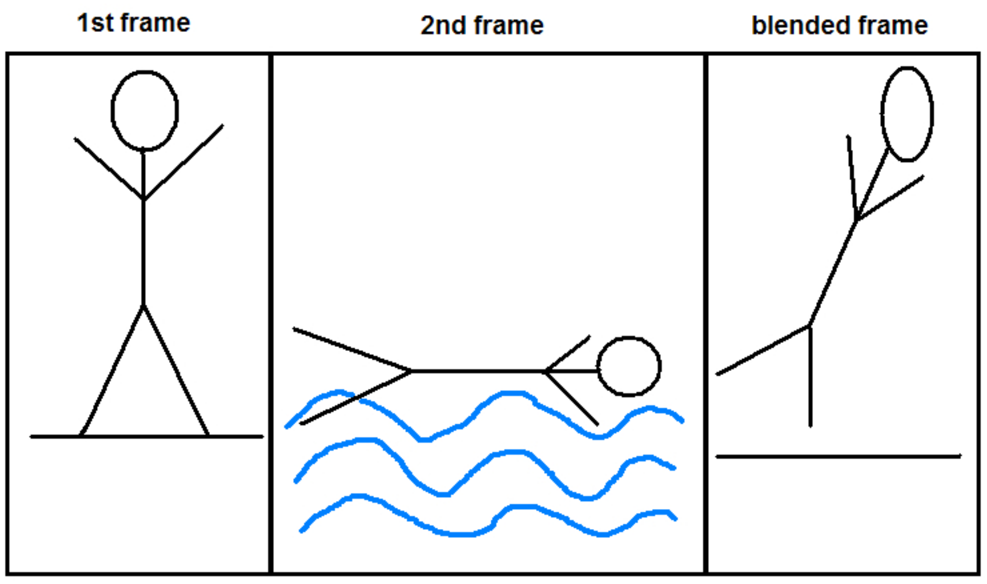


Figure 3: Two frames are blended with weight (= 0.5), result is logically unacceptable

rithms (inverse kinematics, mass distribution, etc.) must be included to get satisfactory blends. We decided to deal with the problem of proper timing of input motions. More complex description of timing can be seen in section 4.5.

4 Registration Curves

In this section, we will describe method from the authors Lucas Kovar and Michael Gleicher[2]. We decided to use it partially, only timewarp curve is modified and implemented (other two curves are described below) because of its effectivity and relative simplicity.
A registration curve is an automatically constructed data structure that encapsulates relationships involving the timing, local coordinate frame, and constraint states of an arbitrary number of input motions. These relationships are used to improve the quality of blended motion and allow blends that were previously beyond the reach of automatic methods.
Building a registration curve consists of the following steps :

1. *Creation of a timewarp curve* - we implement and modify it
2. *Creation of a coordinate alignment curve* - out of scope
3. *Determination of constraint matches* - out of scope

4.1 Creation of a timewarp curve

Building a timewarp curve is the most difficult step from the algorithm of registration curves. Therefore it has the highest computational times. Here is the schedule of forming a timewarp curve :

1. Creation of a grid
2. Finding a minimal-cost path
3. Fitting a smooth monotonic function to the frame correspondances

4.2 Creation of a grid

Using the coordinate-invariant distance function (described in chapter 4.2.1), they create a grid where columns correspond to frames from the first motion, rows correspond to frames from the second motion, and each cell contains the distance between the corresponding pair of frames.

4.2.1 A Coordinate-Invariant Distance Function

They extract neigbourhoods of five frames for each compared frame F1, F2. Next, each frame is converted to a point cloud by attaching markers to the joints of the skeleton, forming two larger point clouds. Finally, minimal sum of squared distances between related points is computed over all rigid 2D transformations of the second point cloud :

D(F1,F2)$=\min_{\Theta,x_0,z_0} \sum_{i=1}^{n} w_i \left\| p_i - T_{\Theta,x_0,z_0} \acute{p_i} \right\|^2$ (3)

where p_i and $\acute{p_i}$ are the i-th points of both point clouds, T_{Θ,x_0,z_0} is a rotation about vertical axis (y) and consecutive translation in the floor (x_0, z_0), w_i is the weight of the individual joints.

Assuming the w_i sum to unity, the optimal solution to Equation (3) is achieved under the following transformation :
we use the shorthand notation $\overline{\alpha} = \sum_{i=1}^{n} w_i \alpha_i$

$$\begin{aligned}
\theta &= \tan^{-1}\left(\frac{\sum_i w_i(x_i z'_i - x'_i z_i) - (\overline{x}\overline{z'} - \overline{x'}\overline{z})}{\sum_i w_i(x_i x'_i + z_i z'_i) - (\overline{x}\overline{x'} + \overline{z}\overline{z'})}\right)\\
x_0 &= \overline{x} - \overline{x'_i}\cos\theta - \overline{z'_i}\sin\theta\\
z_0 &= \overline{z_i} + \overline{x'_i}\sin\theta - \overline{z'_i}\cos\theta
\end{aligned}$$

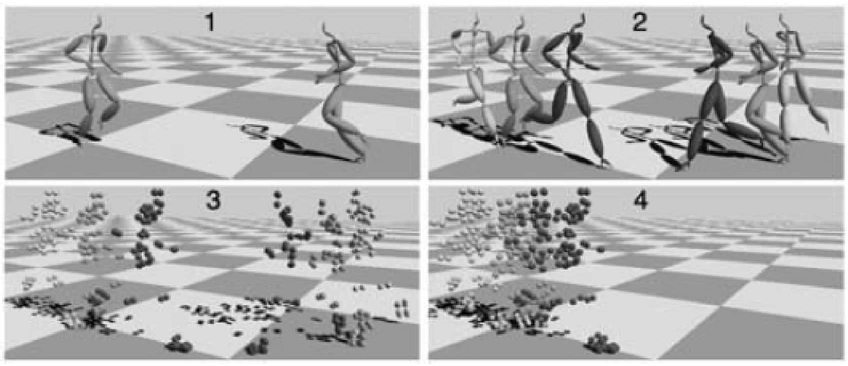


Figure 4: Computing D(F1,F2)[2].

4.3 Finding a minimal-cost path

Given two cells in this grid, they use the dynamic timewarping algorithm to find a minimal-cost connecting path, where the cost of a path is found by summing its cells. From a given starting point, dynamic timewarping is used to find optimal paths leading to all points on the boundary of the grid. However, there is no preferred boundary point. They select the one that minimizes the average cell cost [2]. This path corresponds to an optimal timealignment that starts and ends at the bounding cells. See the example of path in figure 5.

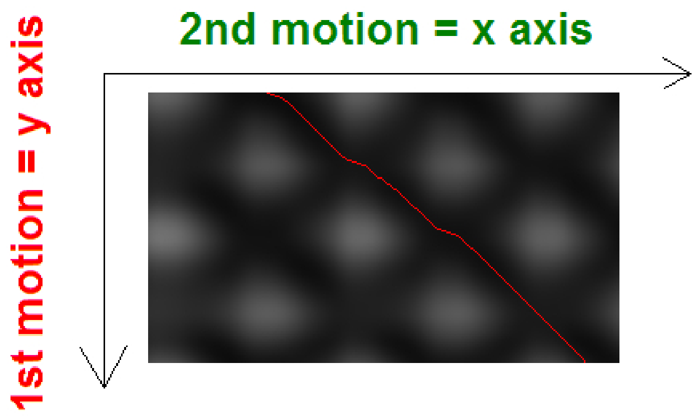


Figure 5: Minimal-cost path, (example genereated by our program Motion Blender), height of the grid equals the number of frames of the 1st motion, width of the grid equals the number of frames of the 2nd motion

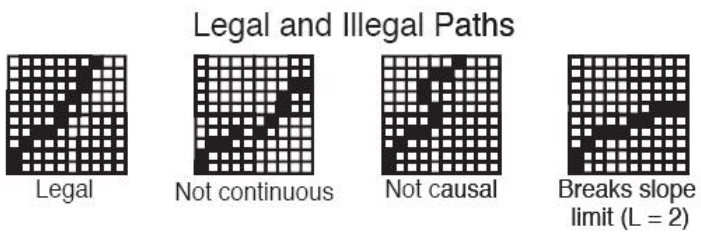


Figure 6: Path musts satisfy these conditions [2].

1. **Continuity**
   Each cell on the path must share a corner or edge with another cell on the path.

2. **Causality**
   Paths must not reverse direction; they should either go entirely forward or entirely backward in time.

3. **Slope Limit**
   At most L consecutive horizontal steps or consecutive vertical steps may be taken.

Slope limits are useful because very large slopes typically indicate that the motions are (at least locally) so different that there are no good frame matches. In this case it is preferable to limit the timewarping and accept the fact that corresponding poses will not be particularly similar. They find a slope limit of 2 or 3 to produce good results [2].

4.4 Fitting a smooth monotonic function to the frame correspondances

They fit a uniform quadratic B-spline as an unconstrained optimization and then adjust the knots. They create the timewarp curve by fitting a smooth, strictly increasing function to the frame correspondences. If we only cared about smoothness, then a simple solution would be to fit a spline, which only requires solving a linear system. Adding in the constraint, that the spline is strictly increasing, produces a significantly more expensive quadratic programming problem. However, given the causality and slope limit restrictions, it is likely that a spline fit without any constraints will be approximately strictly increasing [2].

4.5 Why is timing important

Linear blending does not work in case, when corresponding events occur at different absolute times. Therefore a timewarp curve is built, in order to return sets of frame indices, so that the corresponding frames from each motion match. This algorithm gives better results if the timewarp curve is continuous and strictly increasing. When these conditions are satisfied, the inverse functions for a timewarp curve are defined, and for each frame from input motion they compute corresponding point on the timewarp curve and vice-versa. They align motions so related frames are as similar as possible and then they can average skeletal parameters (see figure 7).

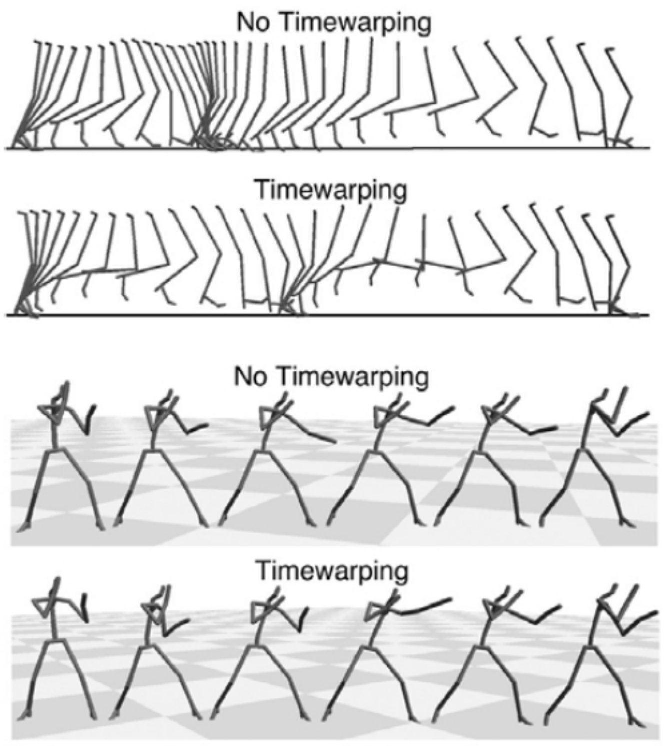


Figure 7: Examples of artefacts when blending without timing [2].

5 Proposed modification of the original algorithm

5.1 Creating the grid

We extract point cloud (see figure 8) for each frame simply by traversing the skeleton structure with particular joint offsets (from hierarchy data) and rotations (from motion data). So, we get absolute coordinates of each joint, which create our point cloud. As in Kovar [2], we use the neighbourhood of five frames to have more information involved. After this extraction for each motion, coordinate-distance invariant function (further distance function) is computed as described in the chapter 4.2.1.

Finally, we create a grid where pixel [i,j] represents the 'distance' (defined in section 4.2.1, Equation (3)) between corresponding frames: i-th from first and j-th from second animation. These are already resampled values from distance function into RGB (0..255), of course. Obtained greyed image gives

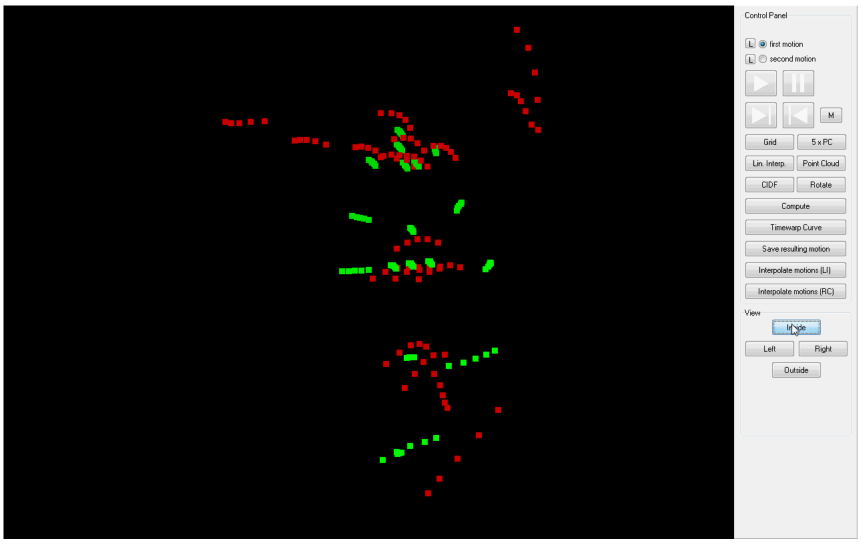


Figure 8: Example of extracted pointclouds in Motion Blender

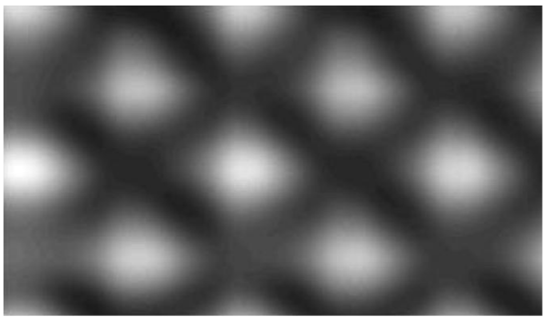


Figure 9: Example of grid of two motions (running vs. walking) 'blendable' with reg. curves, created by our program Motion Blender

us information about the 'relationship' between two blended motions. An image for two motions, satisfying the conditions for our method, is expected to have a cyclic duplicating of some sample. It means that these motions are periodically repeating the same movement (i.e step with first left leg by the walk), and that there are logical correspondencies between the most similar couples of frames (one frame from each motion). Similar couple of frames means that distance function of these frames returns small value. As comparison, we show two grids where on the first image (figure 9), input motions are blendable, however on the second (figure 10) are not. Minimal-cost path is optimal path, because there are the smallest differences between corresponding frames in individual couples (see figure 11 for an example of dissimilar frames).

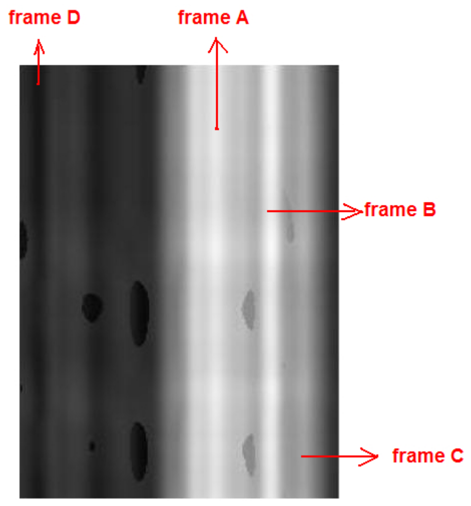


Figure 10: Example of grid of two motions (chopping wood vs. dancing), which are ’unblendable’, in the figure below all four frames are depicted, created by our program Motion Blender

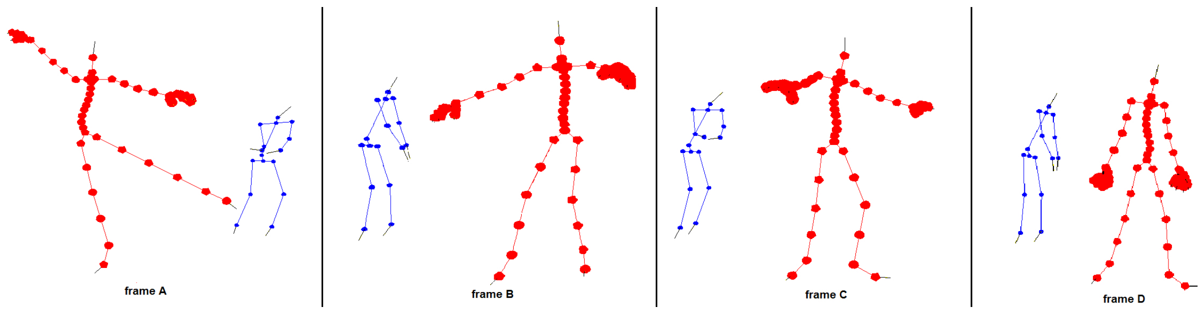


Figure 11: Individual frames, which are selected on the grid, frame D has the most similar poses, other are dissimilar

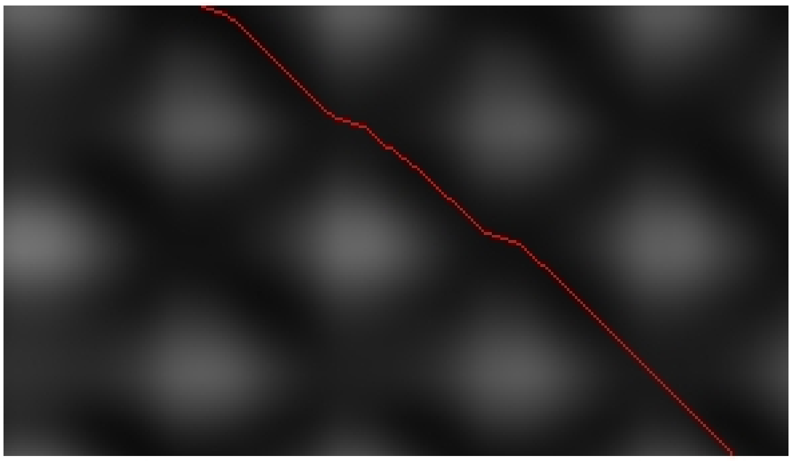


Figure 12: Computed path using brute force recursion

5.2 Minimal-cost path

Given a complete table of distances comparing each frame from the first motion with each from the second, we try to find a minimal-cost path.

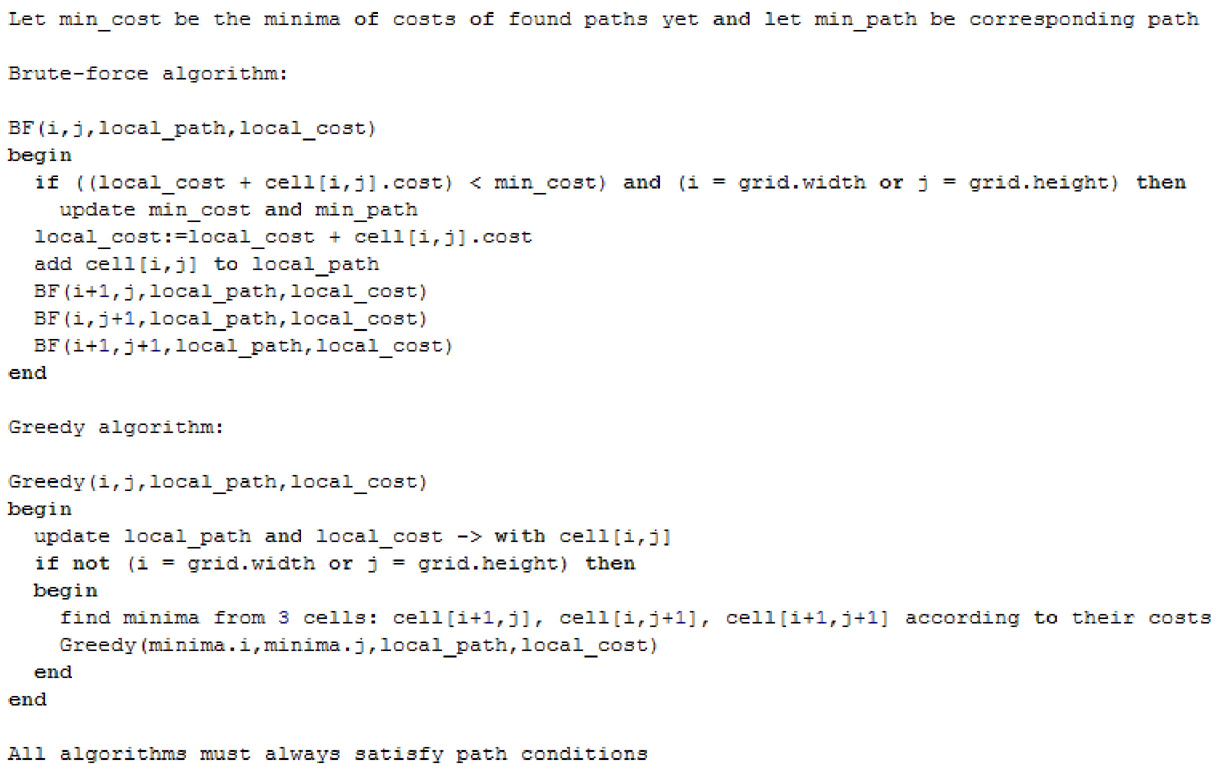
5.2.1 Used methods

1. Brute force algorithm

   This method is based on backtracking, which traverses each possible path, counts the cheapest local path and gives the optimal path (figure 12). Advantage of this approach is that we are sure that there is no lower-cost path, so it always finds the optimal path(in the finite time), on the other hand, this reflects on the computational time.

2. Greedy algorithm

   Greedy algorithm chooses always the best local solution of the problem. We implement it as choosing the cheapest direction (that is allowed because of slope limit) in each recursion nesting. Very fast execution of this algorithm is compensated with the poor quality (means higher cost) of the resulting path, because choosing locally best solution does



```
Let min_cost be the minima of costs of found paths yet and let min_path be corresponding path

Brute-force algorithm:

BF(i,j,local_path,local_cost)
begin
  if ((local_cost + cell[i,j].cost) < min_cost) and (i = grid.width or j = grid.height) then
    update min_cost and min_path
  local_cost:=local_cost + cell[i,j].cost
  add cell[i,j] to local_path
  BF(i+1,j,local_path,local_cost)
  BF(i,j+1,local_path,local_cost)
  BF(i+1,j+1,local_path,local_cost)
end

Greedy algorithm:

Greedy(i,j,local_path,local_cost)
begin
  update local_path and local_cost -> with cell[i,j]
  if not (i = grid.width or j = grid.height) then
  begin
    find minima from 3 cells: cell[i+1,j], cell[i,j+1], cell[i+1,j+1] according to their costs
    Greedy(minima.i,minima.j,local_path,local_cost)
  end
end

All algorithms must always satisfy path conditions
```

Figure 13: Pseudocodes of brute force and greedy algorithms

not always lead to finding an optimal path (figure 14).

3. Recursion with thresholds

   This is the method which we created during testing and verifying former two methods. From experiments, we have determined two optimalizations, appearing to be very effective upgrades of slow, respectively unaccurate methods mentioned above.

   The first optimization includes the fact, that the cheaper paths almost always belong to the shorter ones. We employ this and add a condition which prefers shorter path against the longer when deciding in which direction should the computation follow.

   Without the loss of generality we can assume that we search the path

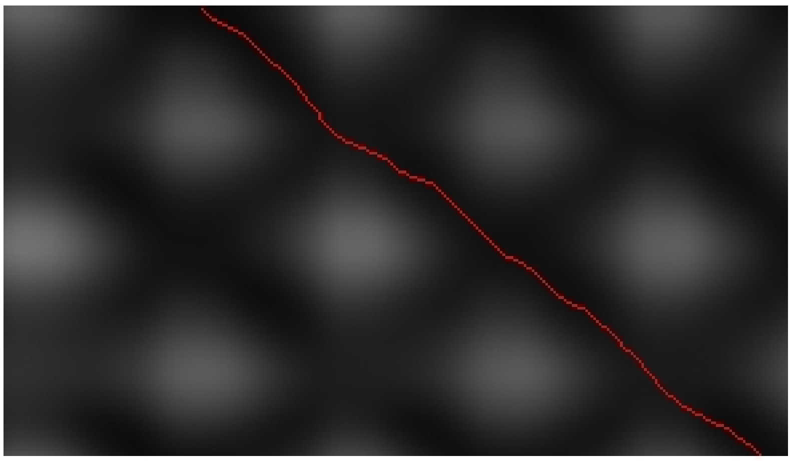


Figure 14: Computed path using greedy algorithm

from the upper-left to the lower-right corner. At the picture below (figure 15) we see the situation when the direction of the red arrow is optimal, because we choose the shortest path and a cell with the smallest value.

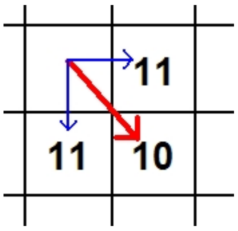


Figure 15: First optimization

The second optimization involves the fact, that optimal path should not contain the cells with high costs, because we want to blend the most similar (which means the cells with small costs) frames together. Therefore we have searched threshold ('area threshold') determining, which cells are too costly and need not to be traversed by recursion. This value is obtained experimentally.
Pseudocode of this algorithm is derived from the brute force algorithm, only our two thresholds (mentioned above) must be incorporated.

5.3 Timewarping with discrete path

Finding a minimal-cost path

Having a grid of two motions, we want to find a minimal cost path from the beginning of motions to their end. This path, leading from one corner to its opposite, represents couples of frames that are blended during blending. It is not necessary to find path exactly between corner pixels, that can be pixels from the close area. We search the first one third of pixels in horizontal and the first one third of pixels in vertical direction and we choose a minima of these values as our beginning of path. This is shown in the picture below (figure 16), where two green lines represent the area of searching the beginning pixel of the path. Then we use our recursive method for finding a minimal-cost path. The implementation of recursion must ensure that the resulting path will satisfy conditions described in the chapter 4.2.1 of this work (depicted at the picture 6). Below, we compare all implemented methods and discuss, which method is the best solution.

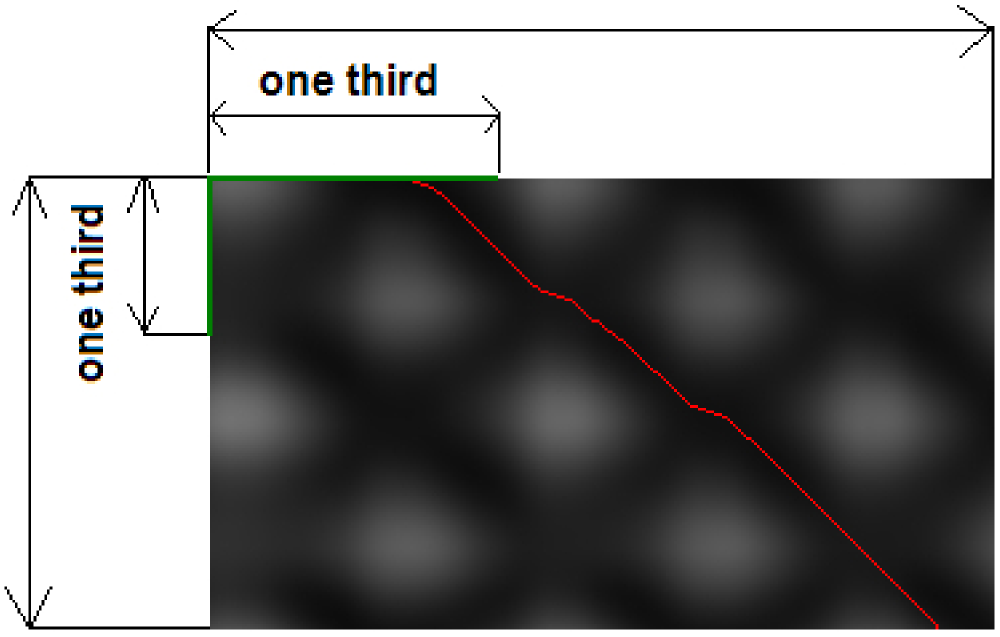


Figure 16: Minimal-cost path (red line), area of possible beginning of path (green lines), selected one third of pixels where we search for the possible starting point of the path

The representation of minimal-cost path

The optimal path is represented as an array of characters, where each character means one pixel. Which character is used depends on the behavior of the path. This information allows us to know the length (number of pixels) of the resulting motion and also direction in each individual step. An example is shown in the figure below (17).

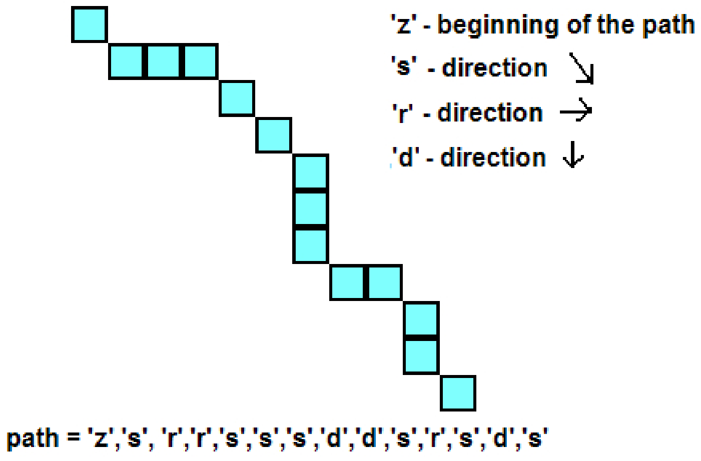

'z' - beginning of the path
's' - direction ↘
'r' - direction →
'd' - direction ↓
path = 'z','s', 'r','r','s','s','s','d','d','s','r','s','d','s'

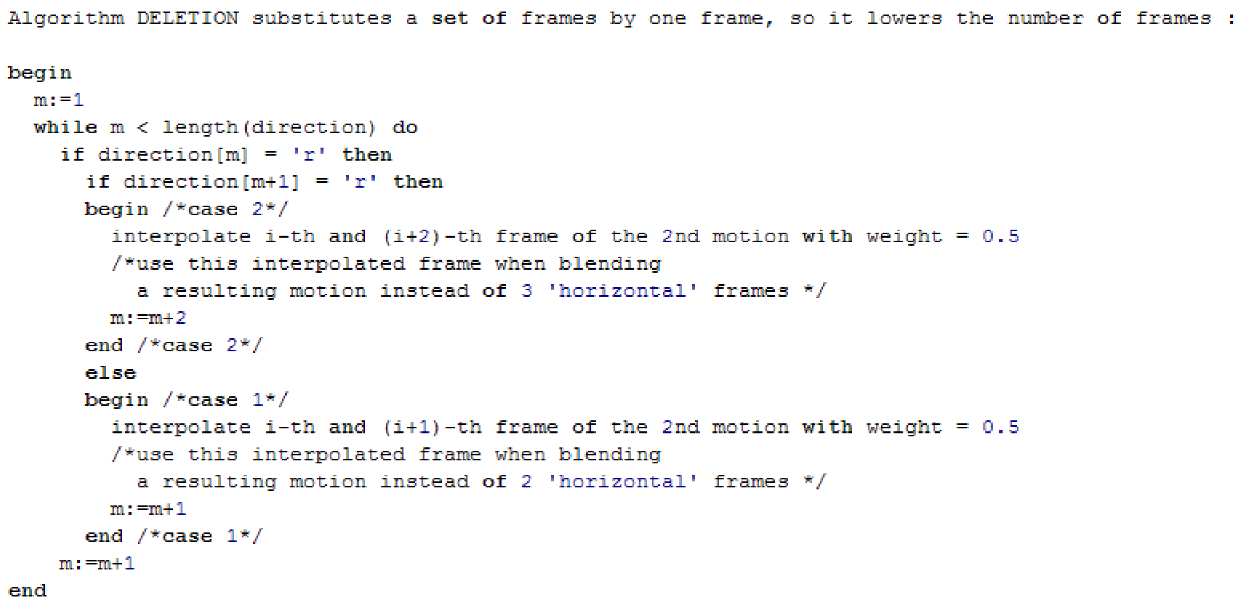
Figure 17: Example of the path

Blending by using a timewarp curve

Although we have a minimal-cost path, the resulting motion after linear blending would not be realistic. The cause is that in our path we allow more than one consecutive pixels in the same (horizontal resp. vertical) direction. This means that a pixel from one motion is consecutively blended with two or three pixels from another motion. This may cause unpleasant deceleration of the resulting motion in those pixels. We get rid of this uncorrectness with the following algorithm.

Corrections of timewarp curve

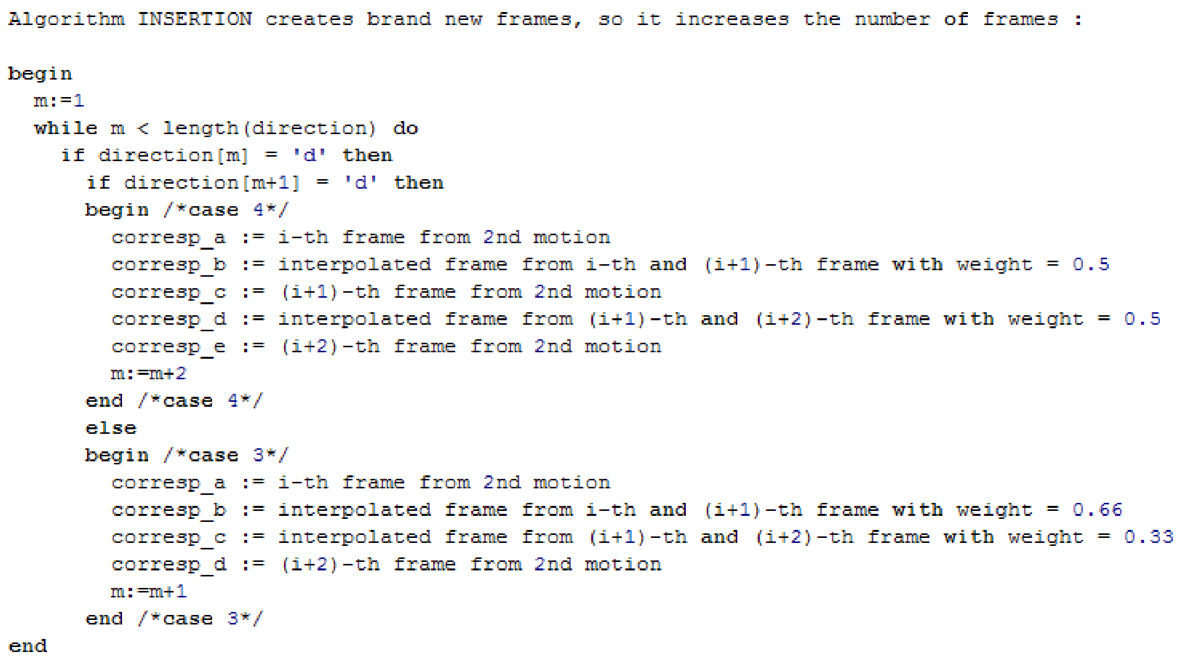
We consider 4 basic cases when the path must be corrected. All of these are depicted in the following scheme (see figure 20). Our main problem is, that we want to find for each frame from the first motion a corresponding frame from the second, according to the minimal path. This corresponding frame from the second motion sometimes does not exist (in cases 3 and 4) or



```
Algorithm DELETION substitutes a set of frames by one frame, so it lowers the number of frames :

begin
  m:=1
  while m < length(direction) do
    if direction[m] = 'r' then
      if direction[m+1] = 'r' then
      begin /*case 2*/
        interpolate i-th and (i+2)-th frame of the 2nd motion with weight = 0.5
        /*use this interpolated frame when blending
          a resulting motion instead of 3 'horizontal' frames */
        m:=m+2
      end /*case 2*/
      else
      begin /*case 1*/
        interpolate i-th and (i+1)-th frame of the 2nd motion with weight = 0.5
        /*use this interpolated frame when blending
          a resulting motion instead of 2 'horizontal' frames */
        m:=m+1
      end /*case 1*/
    m:=m+1
end
```

Figure 18: pseudocode-deletion



```
Algorithm INSERTION creates brand new frames, so it increases the number of frames :

begin
  m:=1
  while m < length(direction) do
    if direction[m] = 'd' then
      if direction[m+1] = 'd' then
      begin /*case 4*/
        corresp_a := i-th frame from 2nd motion
        corresp_b := interpolated frame from i-th and (i+1)-th frame with weight = 0.5
        corresp_c := (i+1)-th frame from 2nd motion
        corresp_d := interpolated frame from (i+1)-th and (i+2)-th frame with weight = 0.5
        corresp_e := (i+2)-th frame from 2nd motion
        m:=m+2
      end /*case 4*/
      else
      begin /*case 3*/
        corresp_a := i-th frame from 2nd motion
        corresp_b := interpolated frame from i-th and (i+1)-th frame with weight = 0.66
        corresp_c := interpolated frame from (i+1)-th and (i+2)-th frame with weight = 0.33
        corresp_d := (i+2)-th frame from 2nd motion
        m:=m+1
      end /*case 3*/
end
```

Figure 19: pseudocode-insertion

	Couples of correspondng frames(1st vs 2nd motion)				
	Pair 1	Pair 2	Pair 3	Pair 4	Pair 5
case 1	1 vs 1	2 vs (2iw3)	3 vs 4		
case 2	1 vs 1	2 vs (2iw4)	3 vs 5		
case 3	1 vs 1	2 vs 2	3 vs (2iw3)	4 vs 3	
case 4	1 vs 1	2 vs (1iw2)	3 vs 2	4 vs (2iw3)	5 vs 3

Table 1: Illustration of corresponding frames

sometimes there are many candidates (in cases 1 and 2). Therefore we use INSERTION and DELETION algorithms.

In the cases 1 and 2 (DELETION), there are many frames from the second motion corresponding to the frame number 2 from the first motion. What we do is linear interpolation between two frames (2 and 3 in case 1, resp. 2 and 4 in case 2), and the resulting frame is then in our algorithm blended with the second frame from the first motion.

In the cases 3 and 4 (INSERTION), there is a similar problem, however we do not have many frames and need one, but we have one frame and we need many. So we interpolate frames 2 and 3 and this new frame is considered as corresponding to the frame 3 from the first motion. Analogue approach is used in the case 4, where frames 1 and 2 are blended together and the resulting frame corresponds to the frame 2 of the first motion. Finally, frames 2 and 3 are interpolated and the outcome is then corresponding to the frame 4 of the first motion. HUH. Maybe, the included table (1), resp. pseudocodes (18 and 19) are clearer. (iw means 'interpolated with')

Now, we have for each frame from the first motion a corresponding frame from the second and we can preform blending.

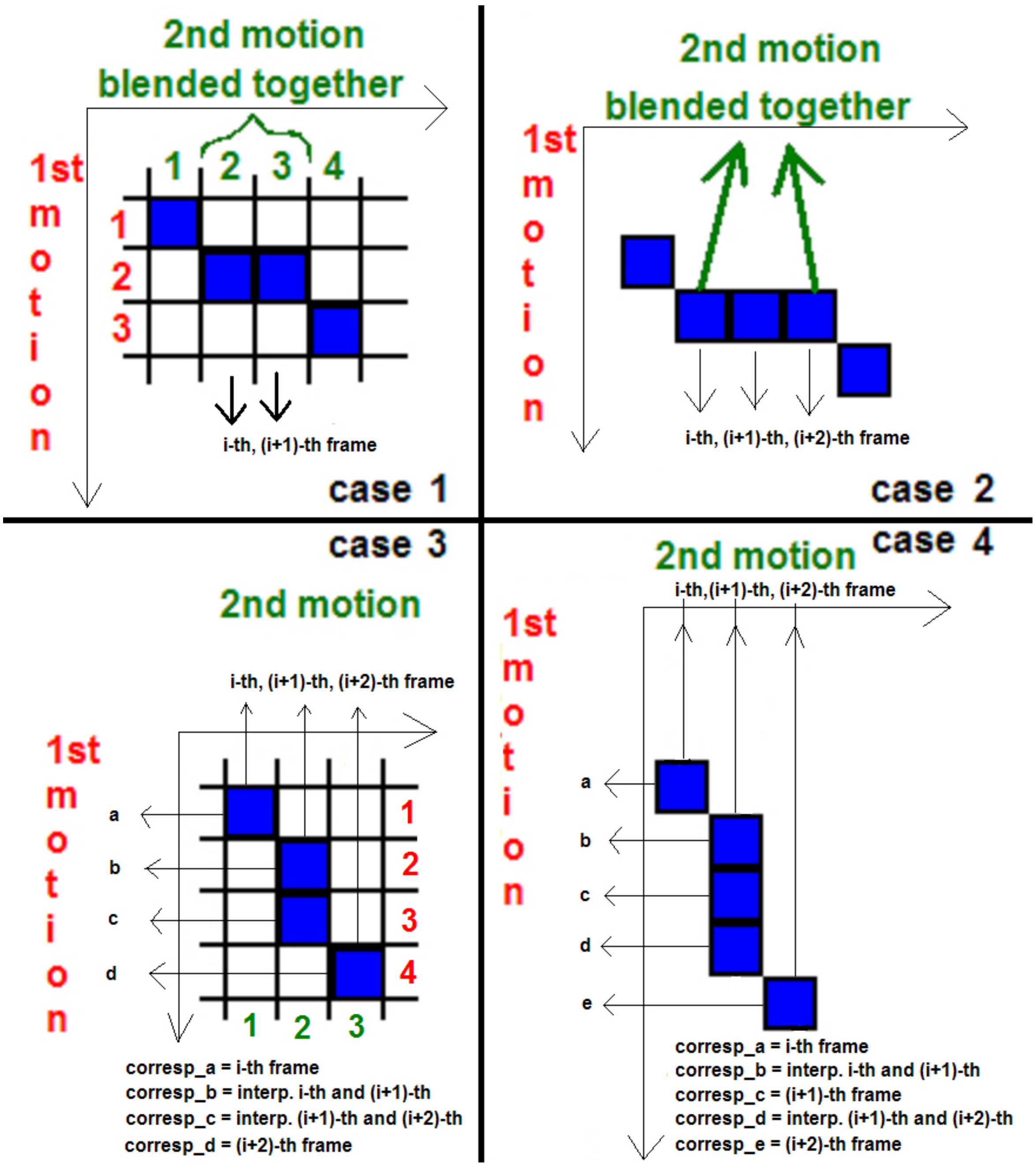

2nd motion
blended together
1st motion
1
2
3
4
1
2
3
i-th, (i+1)-th frame
case 1
2nd motion
blended together
1st motion
i-th, (i+1)-th, (i+2)-th frame
case 2
case 3
2nd motion
i-th, (i+1)-th, (i+2)-th frame
1st motion
a
b
c
d
1
2
3
4
1
2
3
corresp_a = i-th frame
corresp_b = interp. i-th and (i+1)-th
corresp_c = interp. (i+1)-th and (i+2)-th
corresp_d = (i+2)-th frame
case 4
2nd motion
i-th,(i+1)-th, (i+2)-th frame
1st motion
a
b
c
d
e
corresp_a = i-th frame
corresp_b = interp. i-th and (i+1)-th
corresp_c = (i+1)-th frame
corresp_d = interp. (i+1)-th and (i+2)-th
corresp_e = (i+2)-th frame

Figure 20: Corrections of timewarp curve ($corresp_x$ is frame from 2nd motion corresponding to the frame x from the 1st motion)

5.4 Timewarping with B-spline

We incorporate a continuity into solution of blending problem by using a B-spline. This allows us a great control over the motion itself. We use B-spline with a degree of 3 and with a uniform vector of knots. To satisfy the condition, that curve must start at the first point of the path and end at the last point, we must multiply the starting and ending vertex (each two times). So we get a continuous, monotonic curve representing the optimal path and we can obtain a frame corresponding to an arbitrary point of this curve.

Let us have (x,y) coordinates of the point on the curve, these real numbers represent the frames from two input motions, which must be blended to get the frame corresponding to the point [x,y]. Y-coordinate corresponds to the first input motion, however it is a real number so it is bounded by two neighbouring frames(represented by integer values). We gain the frame corresponding to the y-coordinate by interpolation between these two neighbouring frames with a weight that is defined by decimal fraction of the y-value. Analogue computation executes for x-coordinate and second input motion. For clearer understanding see the figure (21).

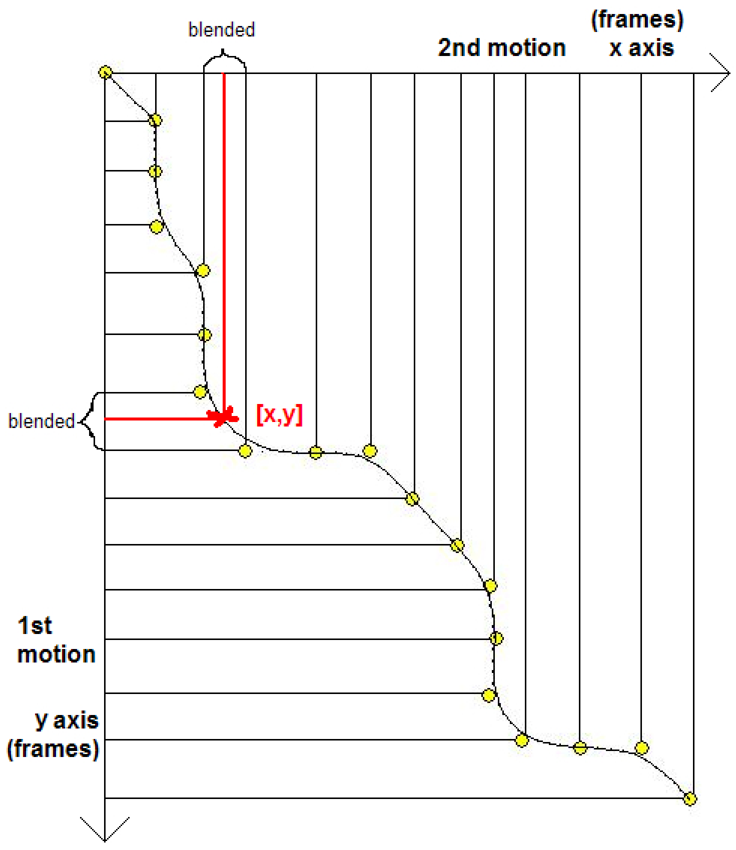


Figure 21: Blending with B-spline

6 Software Implementation

This part deals with software implementation of the presented work. Here we describe program architecture, its features, platforms, used programming languages and implemented user interface.

6.1 Name

Name : Motion Blender

6.2 Our Implementation

6.2.1 BVH Processing

Animation files are loaded by our own loader which supports various types of bvh standards (i.e. 'zxy','zyx',..). On one hand, such a hierarchical data makes preprocessing and implementation more difficult, on the other hand, it speeds up the computations.

Firstly, structure data are parsed and then skeleton structure is built using recursion. We represent each skeleton with the pointer of its root. We use similar recursive method for saving bvh files.
Afterwards, motion part of bvh file is parsed. At first, the proper timing of motion is computed from given frame time and then, motion data is treated.

6.2.2 Linear Blending

This is the simplest method, which blends individual frames in order, as they sequentially follow. While using bvh files where the joint angles (Euler angles) are included and while we are blending motions with the same skeletal structure, we are not forced to use spherical interpolation and instead we use a simple linear interpolation of these angles. Having satisfied the previous conditions, this simplification does not have any visual impact on the resulting blend.

6.3 Information about Program

6.3.1 Purpose

We called our program "Motion Blender" because the main purpose of it is to demonstrate realistic motion blending. Motion Blender allows us to blend human motions and to save the result as standard Biovision file. Blending is performed semi-automatically, the algorithm is visualized and so the user takes a closer look in the theory of human motion representation in computers. Another purpose of Motion Blender is that we can get a brand new high-fidelity motion in a few seconds instead of using motion capture technology.

6.3.2 Features

Motion Blender displays a motion represented by the standard type of Biovision motion file .bvh.
The result of the blending is not only blended motion, but also a grid with optimal path (if found), textfile with individual frame distances and other statistics created by blending algorithm.
In the future, we see possible upgrades of this program in adding other file formats, blending algorithms, making some parametrization of the motion or setting an arbitrary trajectory of moving skeleton.

6.3.3 Input/Output

As mentioned before, user specifies input motions by loading Biovision motion file into the program. We have chosen this file because it has simple structure, it is easy to visualize motion which it represents, and also because it is a standard format and therefore widely supported. Here is its definition :

Bvh-file defines motion of hierarchical skeleton, so the movement of the bone depends on the movement of its predecessor. There is defined the number of frames and the length of frame. Motion is represented by the value of each channel from the hierarchical part. Hierarchical order is important when we want to edit the motion.
.bvh example :

```
HIERARCHY
ROOT Hips
{
        OFFSET 0.00 0.00 0.00
        CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
        JOINT Hip
                {
                OFFSET 0.000000 -17.940001 0.000000
                CHANNELS 3 Zrotation Xrotation Yrotation
                End Site
                        {
                        OFFSET 0.000000 -3.119999 0.000000
                        ...


MOTION
Frames: 20
Frame Time: 0.033333
0.00 39.68 0.00 0.65 ...
```

6.3.4 Understanding bvh structure

A bvh file consists of two parts, a header section describing the hierarchy and initial pose of the skeleton; and a data section containing the motion data. The start of the header section begins with the keyword "HIERARCHY". The following line starts with the keyword "ROOT" followed by the name of the root segment of the hierarchy. After this, hierarchy is described. It is permissable to define another hierarchy, this would be denoted by the keyword "ROOT" too. A bvh file may contain any number of skeleton hierarchies. In practice, the number of segments is limited by the format of the motion section, one sample in time for all segments is on one line of data and this will cause problems for readers which assume a limit to the size of a line in a file.

The world space is defined as a right handed coordinate system with the Y axis as the world up vector.

The motion section begins with the keyword "MOTION". This line is followed by a line representing the number of frames, this line uses the "Frames:" keyword and another value indicating the number of frames that are in the file. On the line after the frames definition is the "Frame Time:" definition, this indicates the sampling rate of the data. In the example BVH file above the sample rate is given as 0.033333, this is 30 frames per second, the usual rate of sampling in a BVH file.

The rest data contains the actual motion data. Each line represents one frame of motion data. The numbers appear in the order of the channel specifications as the skeleton hierarchy is parsed.

6.3.5 Interpretation

To get the position of a segment, firstly, a transformation matrix is created from the local translation and rotation information for that segment. For any joint segment the translation information will be the offset as defined in the hierarchy section. We get the rotation data from the motion section. For the root object, the translation data will be the sum of the offset data and the translation data from the motion section. The BVH format does not account for scales so it isn't necessary to worry about including a scale factor calculation.

There are many ways, how to create the rotation matrix from individual rotation data. Here we describe some of them.

The easiest way to create the rotation matrix is to create 3 individual rotation matrices, one for each axis of rotation. Then concatenate the matrices from left to right Y, X and Z.

Another solution is computing the rotation matrix directly.

Remaining part of interpretation is adding the offset information. It is performed simply by inserting the X,Y and Z translation data into the proper locations of the matrix (4th row resp. column). Once the local transformation is created then concatenate it with the local transformation of its parent, then its grand parent and continue recursively.

6.4 The Structure of Program

Motion Blender is object-oriented program having several classes and using a few graphic and mathematical libraries. It has user friendly API and so the user need not to be trained to use it. Some of these components are described below.

6.4.1 Development Environment

As for a programming language we have chosen Delphi (object oriented Pascal). Rendering is provided by OpenGL. The reason why we have chosen these languages instead of the others is simple. The application is fast, consistent, multiplatform, both languages are well-known and also our familiarity with them is at the highest level.

6.4.2 User Interface

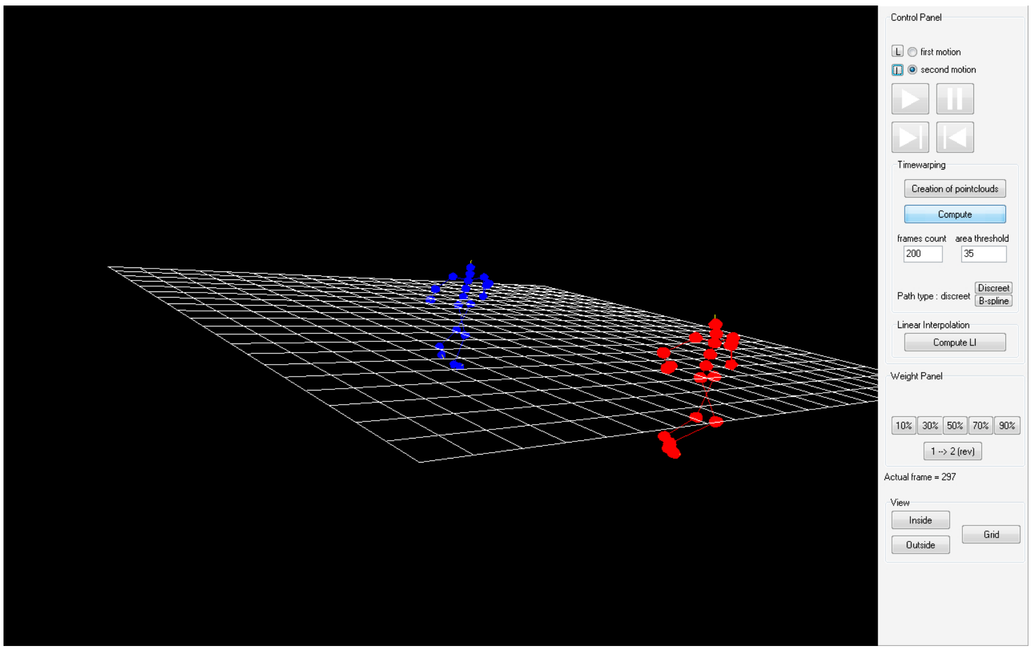
Application consists of scene window, control panel, weight panel and view panel (see figure 22).

Scene window displays the whole scene where all actions are performed. It is a standard 3D OpenGL scene with planar grid, which enables us to visualize the position of objects at the scene. Orientation of the scene is modified by user with the mouse. The skeleton is as simple as possible in order to make the scene continuous and more flexible.

Control panel
User can find there buttons for loading and playing animations (also frame by frame), buttons for creating resulting blends, choosing the used algorithm, fields for setting threshold value and the count of frames of the resulting motion and other important functionality.

Weight panel
Here can user select weight vector that will be used for blending. Program allows blending with constant weights, or creating a transition motion by using dynamic weights.

View panel
In this panel, there are several buttons mostly used for moving the camera in the scene in order to create a better view.



frames count 200

area threshold 35

Path type : discreet

Actual frame = 297

Figure 22: Screen from Motion Blender

6.4.3 Source code overview

Units :

Unit1.pas
- interface between GUI and motion classes (forms, events, buttons)

TFileBVH.pas
- implementation of motion classes, especially parsing of bvh file (hierarchical and data part), loading and saving of bvh files, creation of data structures and using them in order to visualize motion data

MyGLInit.pas
- auxiliary unit with some helpful openGL functions for scene inicialization

MotionBlending.pas
- implementation of blending algorithms, i.e. algebraic classes

Classes :

TJoint

- basic class, creates a node in the skeleton tree, carries information about its name, offset, rotations and pointers to its children and parent
- special method extends the array of descendants

TFrame
- carries information from data part in bvh file, where each line means exactly one frame
- included data represent the channels of joints (mostly 6 decimal places)
- included constructor that performs linear blending of two input frames and arbitrary weight

TAnimation
- except name, it has also an array of frames which correspond to the particular animation

TSkeleton
- carries information about its global position, frame duration (in seconds), pointer to the root, number of joints, currently rendered frame, number of frames and animation object
- special method displays skeleton in actual frame at the global position

TQuaternion
- implemented algebraic object representing one particular rotation

TAxisAngle
- another form of rotation representation

TPointCloud
- point cloud specific for each frame
- data consists of absolute coordinates
- special method displays pointcloud of particular frame

TBlending
- implemented functions for conversion between matrices, quaternions and axis angles
- spherical linear interpolation, weighted average
- computations of aligning values, timewarp curve, etc.

TCell
- the base of recursive traversing the grid
- contains the path from the beginning of the searching

- contains also the cost of this path

6.4.4 Programming features

Pointers
- we implement them because it minimizes memory in use

Recursion
- recursively defined skeleton in bvh file forces us to use this technique, advantage is partial support in OpenGL renderer

Comments
- we use English language

6.4.5 Data structures

text file
- bvh file

tree
- we keep hierarchical skeleton information there (no fixed n-ary tree allowed, because joint can have multiple children)

OpenGL matrix stack
- we use PushMatrix and PopMatrix for recursive joints and bones manipulating and rendering

7 Experiments and Verification

We perform experiments on an AMD Sempron PC (2GHz processor and 1GB memory). As input files we use free bvh files, which we achieved from CMU Graphics Lab Motion Capture Database. While we use always the same set of motions for our experiments, it provides us the possibility to choose, which method gives more plausible and realistic animation. In all our experiments we use constant blending weights(=0.5), in order to simplify verification of the results.

Another criteria for our testing is the robustness of the method. We test which classes of human moves can be synthesized, for which motions it gives a non-realistic results and also for which motions it is impossible to create transition with this algorithm. This is also described in this chapter.

Verification is performed on one hand by computer, who has a few restrictions for the satisfying outcome and on the other side by the human eye, because the computer can not decide if the motion has needed visual form (it is not accomodated to the human motion).
[2]

[2]The data used in this project was obtained from mocap.cs.cmu.edu. The database was created with funding from NSF EIA-0196217.

7.1 One example of blending experiment

We have chosen two input motions : walking (298 frames, 31 joints) and running (173 frames, 31 joints), that were be blended. Blending in our program lasted 18 seconds plus preprocessing which consists of creation of pointclouds. Included graph 23 shows comparation of movement of the left foot in motions. It demonstrates that the pace of the resulting motion is between the paces of the input motions and also that time when the left foot is touching the floor is shorter than by walking and longer than by running. See a grid in 24.

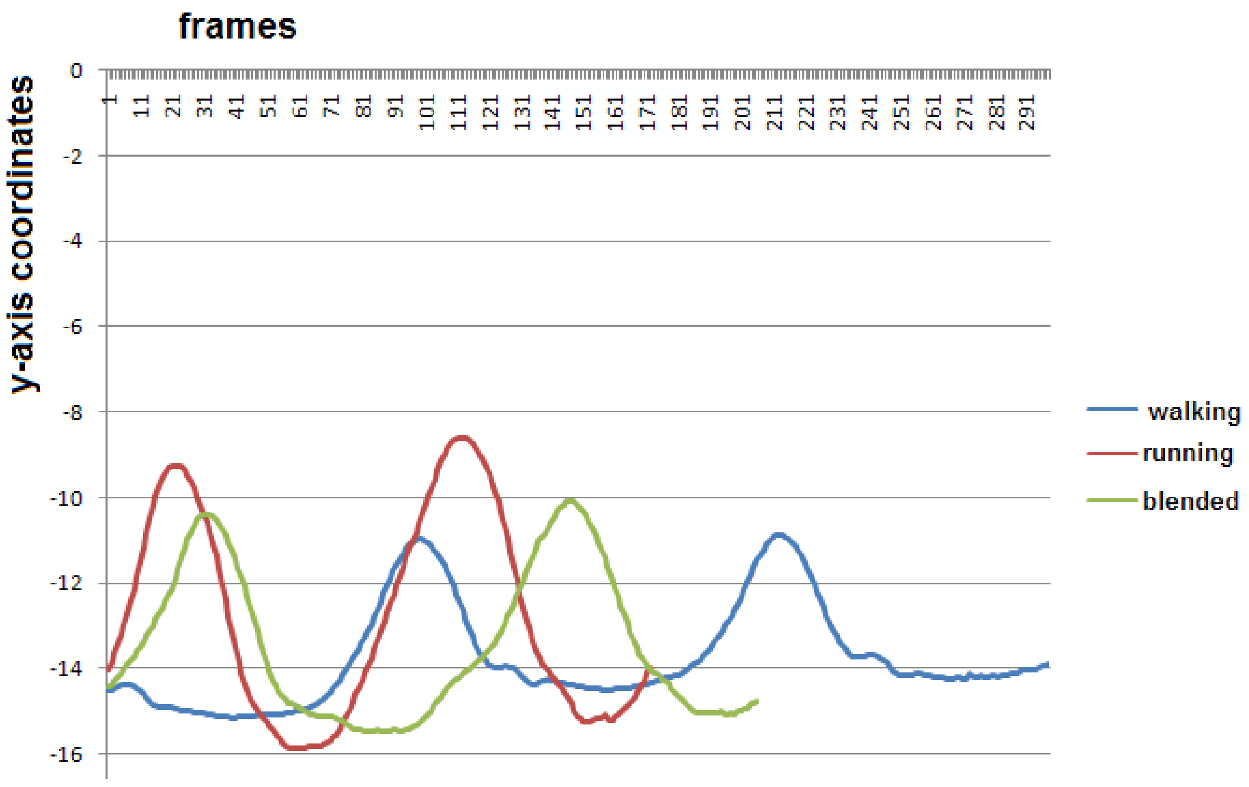


Figure 23: Graph of positions of the left foot in the input motions and in the resulting blended motion

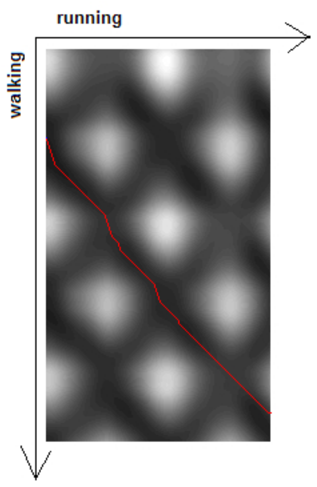


Figure 24: Grid for blending walking and running (constant weight = 0.5)

7.2 Restrictions on the input motions

Unfortunately, there is no blending method, that could blend an arbitrary chosen motions. In this section, we discuss the problem of choosing proper motions which are 'blendable' for our methods.

1. The number of joints

   Input files should be of the same skeletal hierarchy or at least they should have the same number of joints. If we imagine two motions with different skeletons, how could the outcome of blending look like? If the numbers of joints do not equal, our algorithm fails because it is not able to find corresponding joints from each skeleton, and therefore the grid is not created. This condition is fulfilled always when the numbers of joints equal, however, we get the best results in cases when both input files have identical skeleton structure.

2. Different frame rate

   Also frame rate creates the condition of 'blendability'. If we have one motion captured in 20 fps and second in 40 fps, these motions are 'unblendable'. The resulting transition could not look like realistic, because the output would have to be resampled to get the same high fidelity look. There is also much smaller probability of failing to find minimal-cost path, when input motions have the same frame rates. In Motion Blender (our program), the outcome of the blending looks the most realistic when input motions are of the same frame rate.

3. Logical similarity

   This is the most important criteria. Blending algorithms are a great tool for motion synthesizing, however they work only for some subsets of motions. As seen in the figure (10), chopping and dancing motions can not be effectively blended. These subsets are formed by logically similar motions. For example running and walking motions in all paces and directions are proper inputs for blending algorithms. Another subset is formed by kick motions where blending allows us to parametrize the resulting motion and determine the height and the speed of the kick (analogue for punches). Our method works fine with motions from such subsets, because it is based on the similarity of skeleton poses during animation. It does not matter if the motion covers long run or only a few cyclic steps, although the longer input motions, the longer computation of blending.

7.3 Experiments

We have used for experiments (see table 2) of finding minimal-cost path 10 walking and running motions, which have given us some reasonable results(blendable). Column 'Accuracy' shows the ratio of costs of individual path to optimal path. 100 percent accuracy means that resulting path is optimal. Calculation time is measured after preprocessing, which consists of creation of the grid, and the timer is stopped after the path is being found. Accuracy is computed as follows:

Method	**Calculation time**	**Accuracy**
Brute force	81 s	100%
Greedy	$<$ 1 s	92,4%
Optimized recursion	8 s	100%

Table 2: Finding optimal path, counting in integer values (statistics)

Accuracy=100$\times\frac{cost_of_particular_path}{cost_of_optimal_path}$ (4)

And another comparation of two basic methods when we compute in real numbers. The lower table (3) shows, that brute force recursive traversing lasts unacceptable long time, so for computations in real numbers we do not recommend this method.

Method	**Calculation time**	**Accuracy**
Brute force	1179 s	100%
Greedy	$<$ 1 s	94,9%

Table 3: Finding optimal path, counting in real values (statistics)

We also experimented with area threshold (4) and our main task was to find out its best values for blending. We define the desired value as the percentage of the maximum cost of the cell in the grid (with frame distances). Calculation time is measured after preprocessing, which consists of creation of the grid, and the timer is stopped after the resulting blended motion is successfully saved. Threshold area is defined as follows:

threshold_area=100$\times\frac{cost_of_actual_cell}{maximal_cost_of_cell}$ (5)

area value	**Calculation time**	**Accuracy**
20%	failed	failed
30%	21 s	100%
40%	36 s	100%

Table 4: Searching for optimal area threshold (statistics)

Experiments show that interval 25-45 percents satisfies the best our requirements for the speed and accuracy. With values under 25 percents algorithm often fails to find optimal path and when higher values than 45 percents are taken into account, the computation lasts sometimes multiple times longer. We got the best results with area threshold in interval 30-35 percents.

7.4 The comparation of implemented algorithms

We implemented three algorithms for blending and creating transitions into Motion Blender. In this section we will discuss and compare their speed and efficiency.

1. Linear blending

   This is the simplest method, which blends individual frames in order, as they sequentially follow. While using bvh files where the joint angles (Euler angles) are included and while we are blending motions with the same skeletal structure, we are not forced to use spherical interpolation and instead we use a simple linear interpolation of these angles. Having satisfied the previous conditions, this simplification does not have any visual impact on the resulting blend.
   We have experienced that for motions with similar timing and frame poses it gives satisfying blends, however it is only a tiny subset. Because of its simplicity, linear blending has the best time performance, it lasts mostly one second in average(for the motions of approximately 300 frames).

   However we get worse results, when trying to make up transition. As we do not find corresponding couples of frames, timing is not adjusted in this method, therefore the resulting transitions are not realistic and they have many artefacts (i.e. root position is blended incorrectly, many foot slides appear, etc.).

2. Timewarping with discreet path

   Description of the method can be found in the 3rd chapter of this work. When comparing a time performances it appears to be the slowest one, but this disadvantage is compensated with the efficiency. Elapsed time of computations depends on many factors. The more frames the input motions have, the longer the computation lasts. Also the shape of the optimal path determines the execution time. Having a big slope(depicted in the figure 26), many further adjustments must be done, and of course, it prolongs the calculations. Another influencing factor is the area threshold, determining which cells of the table are too high to be a part of the minimal cost path. Setting up this threshold, we are able to dramatically speed up the algorithm, although we risk the possibility of failing to find the timewarping path.

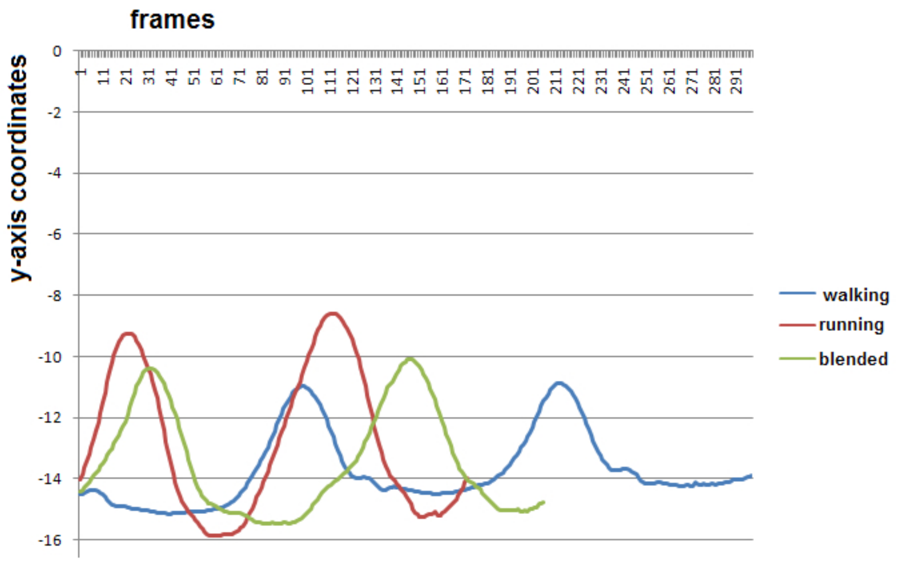


Figure 25: Example of timewarping with discrete path, the pace and minima and maxima of the left foot are blended according to constant weights(=0.5).

Resulting motions do not suffer from the artefacts as in the linear blending method. They have correctly adjusted timing, and blending of corresponding frames ensures high fidelity look. When creating transition or simple blending motions, we are bounded by the frame rate of the input motions, but sometimes it might be usefull to speed up or slow down the animation. This problem is solved by incorporating the B-spline curve.

3. Timewarping with B-spline

As written above, a continuous strictly monotonic B-spline curve is displaced over the discreet points of the minimal-cost path found in the grid. Now we are able to visualize each point of the curve as the frame of the resulting motion. Smoothness of the B-spline ensures that

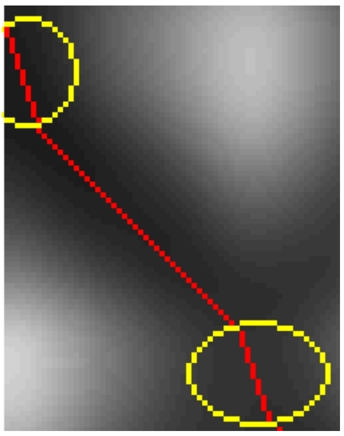


Figure 26: Path in yellow circles has bigger slope than the rest of path

no artefacts from linear blending should occur. Also we can freely resample the resulting motion, simple by adjusting the needed parameter of the curve. Therefore the outputs of Motion Blender are not only blended and transition motions, but also the motions generated at various (arbitrary) frame rates. However, the quality of resulting motion, while upsampling, is low and the movement becomes disconnected. Improvement of this error will be our task in the future.

Time performance is comparable with the second method, elapsed time depends on the number of frames of the resulting motion. The following tests have been performed with these settings : area threshold set as 35, computing a simple blending with weight of 0.5 (see table 5). Time performance is computed as follows:

$$\text{Time\_performance}{=}100\times\frac{time\_timewarping\_discreet\_path}{time\_timewarping\_particular\_method} \quad (6)$$

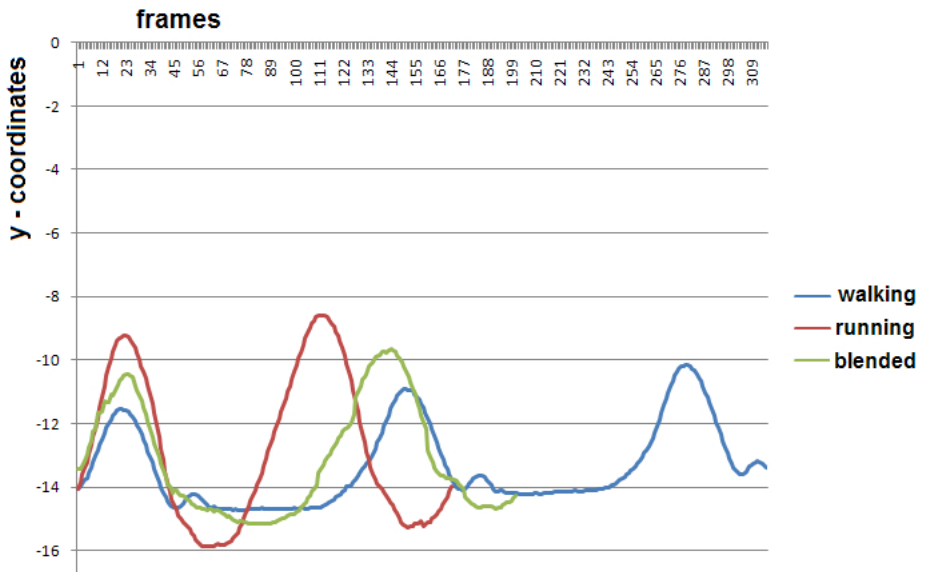


Figure 27: Example of timewarping with B-spline, the resulting curve (green) is not smooth, this is why the motion becomes disonnected. However the pace of motion and the minima and maxima of left foot are blended well (according to our expactations).

Algorithm	**Time performance**
timewarping with discreet path	100%
timewarping with B-spline (output = 200 frames)	88.2%
timewarping with B-spline (output = 400 frames)	100.7%

Table 5: Comparation of computing times for 2nd and 3rd system (with various output settings)

8 Conclusion and Future work

Unfortunately, we have not made out a generally effective blending algorithm (have not been invented yet?). Section with experiments has shown, that we turn each implemented algorithm to advantage in different situations. Linear blending serves the best when used with short motions where no timewarping is needed, then elapsed time of computations is negligible. On the other hand, we recommend to employ timewarping method, when logically related events do not occur at the same absolute times. As we want shorter resulting blend (less than 300 frames) we suggest using B-spline method, in other way discreet path should be used to get the most satisfying outcome.

When comparing time performance with the original paper [2], their creation of timewarp curve lasts 3.43s for two motions of 600 frames, our timewarping with equivalent input motions lasts approximately 10-times longer (same results for discreet path and B-Spline). This may be caused by the fact, that they have used slope limit of 2 (we have used slope limit of 3), slightly differrent approach, another hardware and style of programming may also be factors influencing the results.

We have also succesfully implemented generating transitions, however it is up to the user to specify the length and the placement of the desired transition and to give properly edited input motions. Results are plausible and may be used for further motion synthesis.

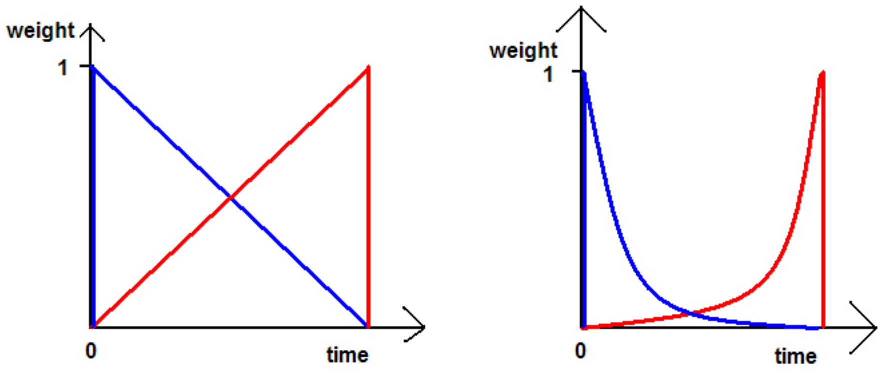


Figure 28: Possible weight distribution during the creation of transition

In this work we have demonstrated the possibilities of motion blending, suggested own improvements and investigated which methods are proper for which inputs. We have presented a framework, which can be used for further synthesis, having such a robust blending capability, we can use it not only for building transitions, but also motion parametrization, creation of motion graphs, continuous motion control, and others.

The quality of transitions could be upgraded by incorporating another type of weight distribution during blending. Our linear distribution and also other possibility can be seen in the figure 28 (our distribution on the left side).

9 Glossary

Retargeting
applying motion data captured from one person to a virtual person of a different size

Inverse Kinematics
the process of computing the pose of a human body from a set of constraints

End effector
joint with no child

Point cloud
set of single points

fps
frames per second

References

[1] S. I. Park, H. J. Shin, and S. Y. Shin. 2002. *On-line locomotion generation based on motion blending.* In Proceedings of ACM SIGGRAPH Symposium on Computer Animation, pages 105-111. 2002.

[2] Kovar L., Gleicher M.. 2003. *Flexible Automatic Motion Blending with Registration Curves.* University of Wisconsin. In Proceedings of ACM SIGGRAPH /Eurographics Symposium on Computer Animation 2003. July 2003.

[3] Perlin K. 1995. *Real time responsive animation with personality.* IEEE Transactions on Visual- ization and Computer Graphics, 1(1):515. 1995.

[4] H. J. Shin, J. Lee, M. Gleicher, and S.Y. Shin. 2001. *Computer puppetry: An importance-based approach.* ACM Transactions On Graphics, 20(2):67-94. Apr.2001.

[5] L. Kovar. 2004. *Automated methods for data-driven synthesis of realistic and controllable human motion.* University of Wisconsin-Madison. 2004.

[6] K. Perlin and A. Goldberg. 1996. *Improv: a system for scripting interactive actors in virtual worlds.* In Proceedings of ACMSIGGRAPH 96, pages 205216. 1996.

[7] D.Winter. 1990. *Biomechanics and Motor Control of Human Movement.* Wiley, New York. 1990.

[8] J. Hodgins,W.Wooten, D. Brogan, and J. OBrien. 1995. *Animating human athletics.* In Proceed- ings of ACM SIGGRAPH 95, Annual Conference Series, pages 7178. 1995.

[9] P. Faloutsos, M. van de Panne, and D. Terzopoulos. 2001. *The virtual stuntman: dynamic characters with a repetoire of autonomous motor skills.* Computers and Graphics, 25(6):933953. 2001.

[10] J. Laszlo, M. van de Panne, and E. Fiume. 1996. *Limit cycle control and its application to the animation of balancing and walking.* In Proceedings of ACM SIGGRAPH 96, Annual Conference Series, pages 155162. 1996.

[11] J. Hodgins and N. Pollard. 1997. *Adapting simulated behaviors for new characters.* In Proceed- ings of ACMSIGGRAPH 97, Annual Conference Series, pages 153162. ACMSIGGRAPH, August 1997.

[12] C. K. Liu and Z. Popovic. 2002. *Synthesis of complex dynamic character motion from simple animations.* ACM Transactions on Graphics, 21(3):408416. 2002.

[13] A. Fang and N. Pollard. 2003. *Efficient synthesis of physically valid human motion.* ACM Trans- actions on Graphics, 22(3):417426. 2003.

[14] M. Neff and E. Fiume. 2002. *Modeling tension and relaxation for computer animation.* In Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2002, pages 8188. July 2002.

[15] A. Bruderlin and L. Williams. 1995. *Motion signal processing.* In Proceedings of ACM SIG- GRAPH 95, Annual Conference Series, pages 97104. ACM SIGGRAPH. 1995.

[16] A. Witkin and Z. Popovic. 1995. *Motion warping.* In Proceedings of ACM SIGGRAPH 95, Annual Conference Series, pages 105108. 1995.

[17] M. Gleicher. 2001. *Motion path editing.* In Proceedings 2001 ACM Symposium on Interactive 3D Graphics. ACM, March 2001.

[18] M. Unuma, K. Anjyo, and T. Tekeuchi. 1995. *Fourier principles for emotion-based human figure animation.* In Proceedings of ACM SIGGRAPH 95, Annual Conference Series, pages 91 96. ACM SIGGRAPH, 1995.

[19] S. Tak, O.-Y. Song, and H.-S. Ko. 2002. *Spacetime sweeping: an interactive dynamic constraints solver.* In Computer Animation 2002, pages 261270. 2002.

[20] Z. Popovic and A.Witkin. 1999. *Physically based motion transformation.* In Proceedings of ACM SIGGRAPH 99, Annual Conference Series, pages 1120. ACM SIGGRAPH, August 1999.

10 Abstrakt

Názov : Syntéza nasnímaných dát pohybu
Autor : Tomáš Sako
Katedra : Katedra aplikovanej informatiky
Dipl. vedúci : RNDr. Stanislav Stanek

Abstrakt: Úvod tejto práce pojednáva o najpoužívanejších a najrozšírenejších metódach syntézy nasnímaných dát pohybu. Prezentujeme tu metódu "časovej krivky" odvodenej od "registračných kriviek" od autorov Lucas Kovar a Michael Gleicher, ktorú sme sami modifikovali. Táto modifikácia by mala urýchliť syntézu dát. Následne ju porovnávame s inými metódami vo vlastnom programe (Motion Blender). Takto meníme pôvodný algoritmus "časovej krivky" zachovávajúci hodnovernosť ľudského pohybu. Výsledkom práce je porovnanie efektivity, rýchlosti a spoľahlivosti implementovaných algoritmov. V záverečnej časti práce je popísaná štruktúra, vlastnosti a použiteľnosť nášho programu a ukazujeme tu výsledky našich experimentov.

Kľúčové slová: nasnímané pohybové dáta, syntéza animácií, skeletálne animácie