



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

2007

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, [month] [year]

Tomáš Sako

Abstract

Title : The synthesis of the motion capture data

Author : Tomáš Sako

Advisor : RNDr. Stanislav Stanek

The introduction of this work deals with the most widely used methods of motion capture data synthesis. Next we present our new method based on dual quaternions, which should speed up the algorithm of data synthesis and 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 upgrade classical Registration curves algorithm by using dual quaternions and get even more effective algorithm that preserves realism of human motion. In the end, we describe the structure, features and usability of the program.

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

Abstrakt

Názov : Syntéza nasnímaných dát pohybu

Autor : Tomáš Sako

Dipl. vedúci : RNDr. Stanislav Stanek

Úvod tejto práce pojednáva o najpoužívanějších a najrozšírenejších metódach syntézy nasnímaných dát pohybu. Prezентujeme tu našu novú metódu založenú na duálnych quaterniónoch, ktorá urýchľuje algoritmus syntézy dát a následne ho porovnávame s inými metódami vo vlastnom programe (Motion Blender). Ide o algoritmus registračných kriviek, ktorý sme doplnili o zmienené duálne quaternióny a získali sme efektívnejší algoritmus zachovávajúci hodnovernosť ľudského pohybu. Výsledkom práce je porovnanie efektivity, rýchlosti a spoľahlivosti algoritmu. V záverečnej časti práce je popísaná štruktúra, vlastnosti a použiteľnosť nášho programu.

Kľúčové slová : motion capture, motion blending, dual quaternions, registration curves, motion synthesis

Contents

1	Introduction	4
2	Basis of Motion Representation	5
	2.1 The Summary of Motion Synthesis Techniques	7
	2.2 Registration Curves	12
3	Software Implementation	16
	3.1 Information about program	16
	3.2 The Structure of Program	19
4	Experiments and Verification	24
5	Conclusion and Future work	25
6	Glossary	26

1 Introduction

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 auxiliary realistic data of 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). My thesis deals with the second problem mentioned, that means realistic synthesis (join) of captured human motions. It is called motion blending.

2 Basis of Motion Representation

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 root) 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.

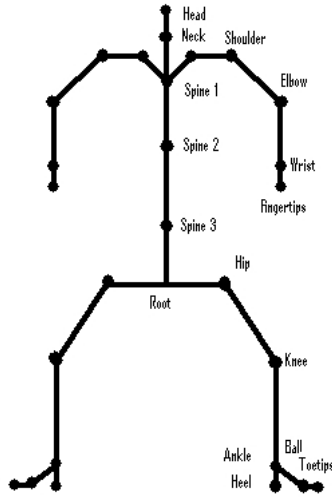


Figure 1: The picture of skeleton

Each motion is represented as the multidimensional function characterizing the skeleton (position and orientation of each joint) at each point in time :

$$M(t) = \left(p(t), q1(t), \dots, qn(t) \right),$$

where p is the position of root in world coordinate system and q_i is the orientation of the i -th joint relative to its parent's coordinate system. These orientations are mostly represented by unit quaternions (also Euler angles possible).

As we know, each motion is defined as a set of frames, where number of frames depends on frame rate. In our motion are frames represented by skele-

ton configurations (poses) $M(t_1), \dots, M(t_j)$ corresponding to regular sampling of our motion $M(t)$. Of course we do not have all frames needed to build particular motion. Frames, that are measured from motion capture, are then interpolated in order to generate in between frames that are missing. We use linear interpolation to generate root positions and spherical linear interpolation (slerp) to generate intermediate joint orientations. Another problem is that human body is not rigid and therefore rough captured data must be pre-processed before their usage in particular blending algorithm. The approximation to the sequence of skeletal poses is made by specific software, but this problem exceeds the submission of this work.

Motion Constraints

Motion constraint is the limitation of some part of skeleton, it is the property that must be fulfilled by motion in particular period. There are different types of constraints i.e. physical constraints (like constraint on head which can not rotate an auxiliary angle) or properties describing the meaning of motion (like hurdles where athlete must jump over each obstacle along the track). While constraints represent the additional information about some period of motion, they can be valid in one frame, many frames or in the whole motion. It is very important to have the possibility to set an auxiliary constraint in motion blending software. It allows us to have great control over the motion and to quickly change particular motion as we like.

2.1 The Summary of Motion Synthesis Techniques

Standard classification of these techniques is : manual methods, physically-based methods and data driven methods.

Manual Synthesis

This is the oldest technique, where the animator specifies individual DOFs (degrees of freedom) 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[9] 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.

Physically-based Synthesis

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[10]. Problem with torques solved Hodgins et al.[11] by using finite state machines and proportional-derivative servos. Also Wooten and Hodgins[12, 13] worked with these methods, they try to generate gymnastic motions. Faloutsos et al.[14] 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.[15] did. Hodgins and Pollard[16] adapted a controller to a new body by computing controller parameters with scaling and consecutive tuning the results using simulated annealing. Faloutsos et al.[17] used support vector machines to compose controllers for different actions. Another method uses a few keyframes and adapts physical laws to motion resulting from simple interpolation. Liu and Popovic[18] 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[19]

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[20] implemented the fact that opposing muscle forces varied the amount of tension. Safonova et al.[21] extracted the main poses, which he used to filter the space of possible character postures. However, there is still no physically simulated method that would provide an arbitrary motion that would be realistic.

Data Driven Synthesis

Invention of motion capture technology has brought realistic example motions of high fidelity, which are used as raw material in data-driven synthesis algorithms. However, example motions can be generated by keyframing or physical simulation. One possibility is signal processing operations applied to each DOF. Bruderlin and Williams[22] introduced some operations like multiresolution filtering, waveshaping and adding smooth displacement maps. Witkin and Popovic[23] proposed motion warping and Gleicher[24] 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. Gleicher[25,26], Lee and Shin[27] and Shin et al.[28] used optimization to coherently adjust DOFs.

There were also some approaches to have a full control of the motion's style and aesthetics. Unuma et al.[29] 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. Chi et al.[30] proposed Laban Movement Analysis to control gestures by controlling their content. Tak et al.[31] checked the position of the body's zero moment point to preserve physical validity. Popovic and Witkin[32] built physically-based framework for editing, that mapped original motion onto a simpler model. Zordan and Hodgins[33] fitted captured data onto physical simulation by proportional-derivative servos to get joint torques that caused individual joint angles. Then it was possible to add some new forces into the scene and to track motion again.

In this category belong also motion blending, motion graphs and parameterizing motions, which will be closely characterized in the next section.

Motion Graphs

Motion graph is very useful tool when we want to create a motion with long duration. It allows us to set a sequence of the orders for the skeleton and create a motion satisfying our specification. We can also reorder the sequence of orders and by using motion graph, we are able to generate new motion very quickly.

Briefly, it is automatic creation of transition motions that seamlessly join different parts of the data. The result is a motion graph where edges represent motion clips and nodes represent generated transitions. Synthesizing motion with a motion graph means to select a sequence of edges, or walk on the graph.

The difficulty of constructing a transition depends on what types of motions will be joined. If two motions are similar then simple interpolation can create a transition. The main task is to find motions, that are reasonably similar and then blend the motions at these points to create transitions.

Parameterizing Motion

Parameterization of motion allows us more effective control over various aspects of motion such as speed, turning angle, and style. It is a tool with which we can edit, setup and recreate a brand new motion of the same fidelity as the initial motion, without the usage of motion capture equipment. For example :

We have motion whose trajectory is described at the picture below. We parametrize it as follows :

Since a motion of constant speed and turning angle traces a circular trajectory, we approximate the root trajectory p of a motion as a circular arc a that best fits the projected trajectory \tilde{p} on the floor.

[1] Let the circular arc a subtend an angle Θ starting from a point a_0 on the circle of radius r centered at o . We find the circular arc a by least-squares fitting which minimizes the distance between \tilde{p} and a as follows:

$$\text{minimize } \sum_{i=1}^{N_f} [\tilde{p}_i - a(\Theta_i, o, a_0, \Theta)]^2 \quad \text{over } o, a_0, \Theta$$

where \tilde{p}_i is the point of the projected root trajectory at the i th frame, $a(\Theta_i)$ is the point on a after rotating a_0 by the angle $(i \times \frac{\Theta}{N_f-1})$ about o , and N_f is the number of the frames. [1] Let l and T be the length of the arc a and the duration of the motion, respectively. Then, the speed of the motion is $\frac{l}{T}$, and its turning angle is $\frac{\Theta}{T}$.

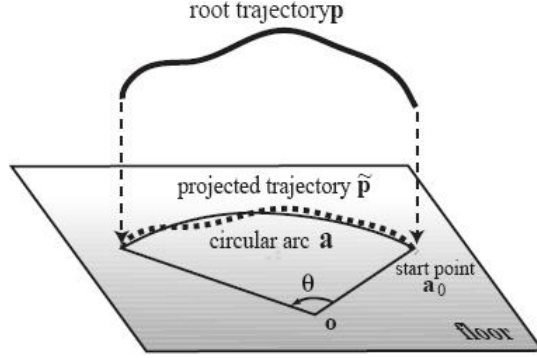


Figure 2: Circular arc as an approximation of the root trajectory [1]

Motion Blending

[1] Basic idea is to generate a motion by blending a set of example motion clips captured from live human motions. First, we need to provide a time-warping scheme to align the motion clips of a wide range of speed. Second, the conventional methods using quaternions (or Euler angles) are required to preprocess the example motions for effective motion blending to ensure that similar poses use similar joint angle representation. When the target character has a different size and proportions from an actor, a blended motion needs to be adapted to the target character.

Retargeting of Motion

Motion retargeting is modifying a motion in order to align it into the proper coordinates of the scene. This is done by blending algorithms and also by animators who want to apply motion to different roles of actors. It means to transform a motion created for one figure to another with the same structure but different segment lengths. Shin et al. [6] suggested an importance-based approach for on-line motion retargeting. They provided the notion of dynamic importance of an end-effector and introduced a fast, robust inverse kinematics solver to realize the important aspect of the end-effector according to its importance value.

Weight Blending

As we have the parameters, we determine the weights of the example motions. Then timewarping is performed to synchronize the example motions. Next, we compute the target motion by blending them with respect to their weights. In the end, having a trajectory and blended postures, we are able to animate the desired locomotion of character, who follows the trajectory

through motion retargeting by using the resulting postures.

Weight computation :

Effective method of this computation is multidimensional scattered data interpolation technique suggested by Sloan et al. [7]. By using cardinal basis functions, Sloan et al. reformulated this scheme to get a more efficient interpolation method. The original version interpolates each degree of freedom per frame, however they computed the weights of the example motions for the given parameter vector and blended them with respect to these weights. For the vector p of parameters, the weight $w_i(p)$ of the i th example motion is defined as

$$w_i(p) = \sum_{l=0}^{N_p} a_{il} A_l(p) + \sum_{j=1}^{N_e} r_{ij} R_j(p)$$

where $A_l(p)$ and a_{il} are the linear basis functions and their coefficients, $R_j(p)$ and r_{ij} are the radial basis functions and their coefficients, respectively.

Posture Blending with Vector Spaces

To interpolate two unit quaternions we use slerp (spherical linear interpolation). Lee and Shin [4] proposed a general framework for constructing the time-domain filters for orientation data. Their scheme satisfies such important filter properties as coordinate-invariance, time-invariance, and symmetry. Buss and Fillmore [5] provided a method for computing the weighted spherical averages of sample points on d -dimensional sphere based on least squares minimization.

There are many algorithms and mathematical approaches which solve the root and joint blending. However, orientation data such as root orientations and joint angles are not so easy to blend. Because of the non-linearity of the orientation space, this method can not be applied directly.

The main idea for blending orientations is transforming the quaternions into their analogues in a vector space with respect to a reference orientation, computing their weighted sum, and then transforming the result back to the orientation space. This ensures us much smaller errors and artifacts in the resulting motion. The connection between quaternions and vectors are the logarithm and exponential maps.

2.2 Registration Curves

In this section, we will describe method from the authors Lucas Kovar and Michael Gleicher[2]. We decided to implement it 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.

A registration curve consists of a *timewarp curve*, a *coordinate alignment curve*, and a set of *constraint matches*.

Linear Blending

Blended ith frame is the result of linear interpolation of the root and spherical interpolation of the joint angles in corresponding frames in each motion involved. It gives reasonable results only for the short blends of the similar motions (small jump and high jump). This simple method does not take constraints into account.

Timing

Linear blending does not work in case, when corresponding events occur at different absolute times. Therefore a timewarp curve $S(u)$ 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 $u(S_i)$ 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.

Coordinate Frame Alignment

Motion itself does not change when rigid 2D transformations are applied. They rotate and translate input motions and get resulting motions that have local coordinate systems as similar as possible. Then they blend related frames from each motion according to the blend weights. An alignment

curve $A(u)$ consists of set of transformations that align the frames at each point of $S(u)$.

Constraint Matches

Pure frame combining need not to satisfy our expectations of the resulting motion. Especially, if the input motions have several constraints, which are not taken into account during the computations. In comparison, walking and running have different types of constraints (contacts with the floor). In this algorithm they presume that constraints at roughly identical absolute intervals mean the same constraint. The start and end of the constraints in global time are parameters and they blend them.

Constructing Registration Curves

They start the construction with a timewarp curve $S(u)$. Another step is building an alignment curve $A(u)$ around $S(u)$, and finally the constraint matches are found (mapped into global time). They use a function $D(F1, F2)$ to determine the "distance" between frames.

A Coordinate-Invariant Distance Function

They extract neighbourhoods of five frames for each compared frame $F1, F2$. Then each frame is converted to a point cloud (points have the same position as the joints). Finally, minimal sum of squared distances between related points is computed over all rigid 2D transformations of the second point cloud.

$$D(F1, F2) = \min_{\Theta, x_0, z_0} \sum_{i=1}^n w_i \|p_i - T_{\Theta, x_0, z_0} p'_i\|$$

p_i and 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.

Creating the Timewarp Curve

They create the timewarp curve from a set of frame correspondences, where a strictly increasing spline is being fit. Then a distance function is used to fill a grid where frames from the first and second motion are compared each other.

Each cell represents a distance between corresponding frames. Afterwards, dynamic timewarping algorithm is applied in order to find a minimal-cost connecting path and this path is optimal timealignment starting and ending at the bounding cells. Following properties should be satisfied : continuity, clausality and slope limit.

Creating the Alignment Curve

Each corresponding frames have specific rigid 2D transformation $\Theta_i, x_{0_i}, z_{0_i}$ that aligns the second frame to the first frame. They fit a 3D quadratic spline to these transformations and the result is an alignment curve

$$A(u) = (\text{identity transformation}(A_1(u) = I), (A_2(u) = \Theta(u), x_0(u), z_0(u)))$$

Identifying Constraint Matches

They map the interval of each constraint into standard time frame via the timewarp curve, and then create constraint matches according to :

1. Each constraint match must consist of exactly one constraint from each motion
2. Union of all constraint intervals must create a single continuous interval in each constraint match

Blending With Registration Curves

Creating a single frame $B(t_i)$:

1. Determine a position $S(u_i)$ on the timewarp curve
2. Position and orient the frames at $S(u_i)$.
3. Combine the frames based on the blending weights $w(t_i)$.
4. Determine the constraints on the resulting frame

Advancing Along the Timewarp Curve

For advancing Δt units of time, they pick $\Delta u = u_i - u_{i-1} - 1$ such that $S_j(u_i - 1 + \Delta u) - S_j(u_{i-1} - 1) = \Delta t$. Hence $\frac{du}{dt} = \frac{du}{dS_j}$. For general $w(t)$:

$$\frac{du}{dt} = \sum_{j=1}^k w_j(t) \frac{du}{dS_j}$$

Positioning and Orienting Frames

Extracted frame $M_j(S_j(u_i))$ from the j -th motion is transformed by $A_j(u_i)$. Now they try to find rigid 2D transformation $T(t_i)$, such that $T(t_i)A_j(u_i)$ would be the total transformation applied to $M_j(S_j(u_i))$. They average votes for $T(t_i)$ from each motion according to the blend weights. For averaging they use $\Delta T(t_i)$ represented as parameter set $\Theta_j, (x_j, z_j)$.

Hence $T(t_i) = \left\{ \sum_j w_j \Theta_j, (\sum_j w_j x_j, \sum_j w_j z_j) \right\}$

Making the Blended Frame

From transformed frames they compute the resulting frame from weighed average of the root positions and joint orientations. Similarly they compute

$$IM = [\sum w_i C_i^s, \sum w_i C_i^e]$$

where $[C_i^s, C_i^e]$ is interval where i -th constraint C_i of constraint match M is valid. When u_i is inside this interval, C_i is included to the blend frame.

Transitions

Firstly, they set the length of the transition (parameter h is the half length) and two frames $M_1(f_1), M_2(f_2)$ from each motion. These frames are in the center of the transition. Obviously, the entire length of the transition is then set to $2h+1$ frames. Now they use dynamic timewarping on $M_1(f_1)$ and $M_2(f_2)$. They compute

$$u_0 = \frac{1}{2}(u(S_1(f_1)) + u(S_2(f_2)))$$

from $M_1(f_1), M_2(f_2)$ to find the beginning $S(u_0)$ of the transition. Consequently, h frames in backward direction and h frames in forward direction are computed using blending weights from $(0,1)$ to $(1,0)$.

3 Software Implementation

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

3.1 Information about program

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 arbitrary human motions and to save the result as standard Biovision file. Blending is performed semi-automatically, the user can edit the resulting motion and so he 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.

Features

Motion Blender displays a motion represented by the standard types of Biovision motion files like .bvh/.bva .

One of the most important features is the comparison of two blending algorithms, standard version of registration curves and our upgraded version with dual quaternions. The result of the blending is not only blended motion, but also a graph, a few tables and other statistics created by blending algorithm. Program allows adding some small changes by editing in each frame, to make specific motion.

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.

Input/Output

As mentioned before, user specifies input motions by loading Biovision motion files into the program. We have chosen these files because they have simple structure, it is easy to visualize motions which they represent, and also because they are standard format and therefore widely supported. There are two types of these files :

Bva-file is the most simple file format. At each frame there are defined

nine values representing translation, rotation and scaling for each bone.

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 for 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 3.430000 0.000000 0.000000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Knee
    {
      OFFSET 0.000000 -18.469999 0.000000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT Ankle
      {
        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 ...

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. Examine the example bvh file above. 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 to be defined. After this hierarchy is described it is permissible to define another hierarchy, this too would be denoted by the keyword "ROOT". 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. You will typically find that BVH skeletal segments are aligned along the Y or negative Y axis (since the characters often have a zero pose where the character stands straight up with the arms straight down to the side).

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 a number 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 a 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 was parsed.

Interpretation

To get the position of a segment you first create a transformation matrix from the local translation and rotation information for that segment. For any joint segment the translation information will simply 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

doesn't account for scales so it isn't necessary to worry about including a scale factor calculation.

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.

$$vR = vYXZ$$

Another solution is computing the rotation matrix directly.

Remaining part of interpretation is adding the offset information. Poke the X,Y and Z translation data into the proper locations of the matrix. Once the local transformation is created then concatenate it with the local transformation of its parent, then its grand parent, and so on.

$$vM = vMchildMparentMgrandparent$$

Also the motion, resulting from blending algorithm, has hierarchical definition. The user has again the choice of Biovision file formats to save his work.

3.2 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.

Development Environment

As for a programming language we choose 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 higher level.

User Interface

Application window consists of scene window, auxiliary window, stats window, property panel, tool panel and main menu.

Scene window displays the particular motion of the skeleton. It is a standard 3D OpenGL scene with 3 coordinate arrows and with planar grid representing the ground. Orientation of the scene is modified by user with the mouse. With the mouse can user also select the joints at the skeleton, while editing the motion. The skeleton is as simple as possible in order to make the scene continuous and more flexible.

Auxiliary window helps user to select joints in case when particular joint is hidden by another one and it is difficult to select it directly from the scene. This window displays static skeleton with all joints, and the selection of some joint also means the selection of that joint at the scene.

Stats window displays detailed statistics of current blending. Not only tables, but also histograms and other important data are presented here.

Property panel is situated at the bottom of the application window. Main application properties are depicted there, such as actual frame, coordinates of the mouse cursor, selected joint, status of blending process, etc.

Tool panel is situated to the left side of the application window. There are control buttons, which adjust the scene and motion properties used for editing and viewing the motion.

Main menu like in the other programs offers the whole functionality of the program, including open, save buttons, buttons for blending statistics overview and other.

Model

Sketch of the class diagram of Motion Blender :

Source code overview

Units :

Unit1.pas

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

TFileBVH.pas

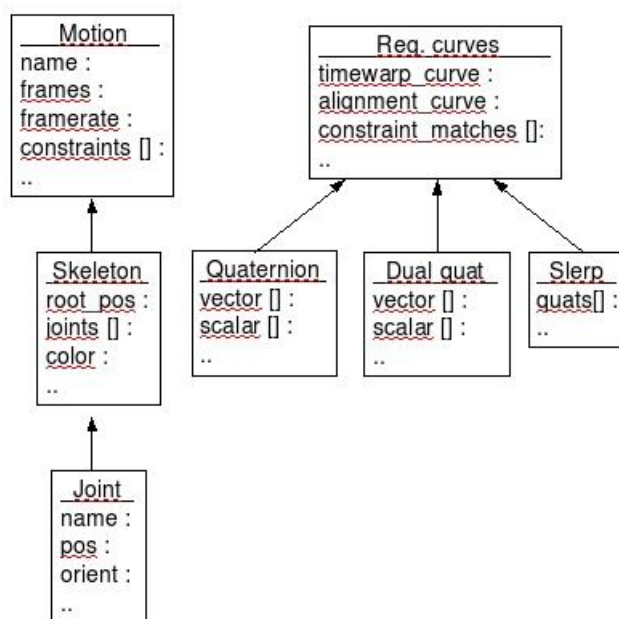


Figure 3: Class Diagram

- implementation of motion classes, especially parsing of bvh file (hierarchical and data part), creating of data structures and using them in order to visualize motion data

MyGLInit.pas

- auxiliary unit with some helpful OpenGL functions for scene initialization

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).

TAnimation

- except name, it has also an array of frames which correspond to the par-

ticular animation

TSkeleton

- carries information about its global position, frame duration (in sec.), 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

Programming features

Pointers

- we implement them and 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

Data structures

text file

- bvh file

common tree

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

OpenGL matrix stack

- we use PushMatrix and PopMatrix for recursive joints and bones rendering

Encountered problems

1. In bvh-files occur inconsistent numeric data separators, the most of files use space, but we have met also with tab as separator.

Solution :

we take the first character which succeeds the 'OFFSET' at the 4th line in bvh file, this character represents the separator used in the whole file.

2. Applying parent's transformations before actual joint's transformations in recursive rendering process.

Solution :

we use PushMatrix and PopMatrix in the proper order (method display in TSkeleton class)

3. It is necessary to apply firstly the offset and secondly rotations at individual joint. It has been a problem in our recursion.

Solution :

We send parent's rotations as parameters and sending three nulls for initial rotations ensure that the offset executes allways before rotations at individual joint.

4 Experiments and Verification

We perform experiments on an AMD Athlon PC (2GHz processor and 1GB memory). As input files we use free bvh files, which were used also in other similar works for testing and experiments. This allows us to compare time performance of registration curves with some other algorithms, which were tested in the past. Thus the identical experimental data provide us the possibility to choose, which method gives more plausible and realistic animation.

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.

Verification is performed 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).

5 Conclusion and Future work

6 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

Bibliography

- [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. 2003.
- [3] PERLIN K. 1995. *Real time responsive animation with personality*. IEEE Transactions on Visualization and Computer Graphics, 1(1):515. 1995.
- [4] J. LEE AND S. Y. SHIN. 2001. *General construction of time-domain filters for orientation data*. IEEE Transactions on Visualization and Computer Graphics. 2001.
- [5] S. R. BUSS AND J. P. FILLMORE. 2001. *Spherical averages and applications to spherical splines and interpolation*. ACM Transactions On Graphics. 2001.
- [6] 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.
- [7] P. SLOAN, C. F. ROSE, AND M. F. COHEN. 2001. *Shape by example*. In Proceedings of 2001 ACM Symposium on Interactive 3D Graphics, pages 135-144. 2001.
- [8] L. KOVAR. 2004. *Automated methods for data-driven synthesis of realistic and controllable human motion*. University of Wisconsin-Madison. 2004.
- [9] K. PERLIN AND A. GOLDBERG. 1996. *Improv: a system for scripting interactive actors in virtual worlds*. In Proceedings of ACM SIGGRAPH 96, pages 205-216. 1996.

- [10] D. WINTER. 1990. *Biomechanics and Motor Control of Human Movement*. Wiley, New York. 1990.
- [11] J. HODGINS, W. WOOTEN, D. BROGAN, AND J. OBRIEN. 1995. *Animating human athletics*. In Proceedings of ACM SIGGRAPH 95, Annual Conference Series, pages 7178. 1995.
- [12] W. WOOTEN AND J. HODGINS. 1995. *Dynamic simulation of human diving*. In Proceedings of Graphics Interface (GI95), pages 19. 1995.
- [13] W. WOOTEN AND J. HODGINS. 2000. *Simulating leaping, tumbling, landing, and balancing humans*. In IEEE International Conference on Robotics and Animation, volume 1, pages 656662. 2000.
- [14] P. FALOUTSOS, M. VAN DE PANNE, AND D. TERZOPOULOS. 2001. *The virtual stuntman: dynamic characters with a repertoire of autonomous motor skills*. Computers and Graphics, 25(6):933953. 2001.
- [15] 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.
- [16] J. HODGINS AND N. POLLARD. 1997. *Adapting simulated behaviors for new characters*. In Proceedings of ACM SIGGRAPH 97, Annual Conference Series, pages 153162. ACM SIGGRAPH, August 1997.
- [17] P. FALOUTSOS, M. VAN DE PANNE, AND D. TERZOPOULOS. 2001. *Composable controllers for physics-based character animation*. In Proceedings of ACM SIGGRAPH 2001, Annual Conference Series, pages 251260. ACM SIGGRAPH, July 2001.
- [18] C. K. LIU AND Z. POPOVIC. 2002. *Synthesis of complex dynamic character motion from simple animations*. ACM Transactions on Graphics, 21(3):408416. 2002.
- [19] A. FANG AND N. POLLARD. 2003. *Efficient synthesis of physically valid human motion*. ACM Transactions on Graphics, 22(3):417426. 2003.
- [20] 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.

- [21] A. SAFONOVA, J. HODGINS, AND N. POLLARD. 2004. *Synthesizing physically realistic motion in lowdimensional, behavior-specific spaces*. ACM Transactions on Graphics, 23(3). 2004.
- [22] A. BRUDERLIN AND L. WILLIAMS. 1995. *Motion signal processing*. In Proceedings of ACM SIG- GRAPH 95, Annual Conference Series, pages 97104. ACM SIGGRAPH. 1995.
- [23] A. WITKIN AND Z. POPOVIC. 1995. *Motion warping*. In Proceedings of ACM SIGGRAPH 95, Annual Conference Series, pages 105108. 1995.
- [24] M. GLEICHER. 2001. *Motion path editing*. In Proceedings 2001 ACM Symposium on Interactive 3D Graphics. ACM, March 2001.
- [25] M. GLEICHER. 1997. *Motion editing with spacetime constraints*. In Michael Cohen and David Zeltzer, editors, Proceedings 1997 Symposium on Interactive 3D Graphics, pages 139148. 1997.
- [26] M. GLEICHER. 1998. *Retargeting motion to new characters*. In Proceedings Of ACM SIGGRAPH 98, Annual Conference Series, pages 3342. 1998.
- [27] J. LEE AND S. Y. SHIN. 1999. *A hierarchical approach to interactive motion editing for human-like figures*. In Proceedings of ACM SIG- GRAPH 99, Annual Conference Series, pages 3948. August 1999.
- [28] H. J. SHIN, J. LEE, M. GLEICHER, AND S. Y. SHIN. 2001. *Computer puppetry: an importance-based approach*. ACM Transactions on Graphics, 20(2):6794. 2001.
- [29] M. UNUMA, K. ANJYO, AND T. TEKEUCHI. 1995. *Fourier principles for emotion-based human figure animation*. In Proceedings of ACM SIG- GRAPH 95, Annual Conference Series, pages 91 96. ACM SIGGRAPH, 1995.
- [30] D. CHI, M. COSTA, L. ZHAO, AND N. BADLER. 2000. *The emote model for effort and shape*. In Proceedings of ACM SIGGRAPH 2000, Annual Conference Series, pages 173182. ACM SIGGRAPH, August 2000.
- [31] S. TAK, O.-Y. SONG, AND H.-S. KO. 2002. *Spacetime sweeping: an interactive dynamic constraints solver*. In Computer Animation 2002, pages 261270. 2002.

- [32] 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.
 - [33] V. ZORDAN AND J. HODGINS. 2002. *Motion capture-driven simulations that hit and react*. In Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2002. 2002.
-
-