Animating human model in OpenGL using data from Vicon system

Gediminas Mazrimas
Aalborg University Copenhagen
Computer Vision and Graphics
g.mazrimas@gmail.com

Algirdas Beinaravicius
Aalborg University Copenhagen
Computer Vision and Graphics
algirdux@gmail.com

Abstract

This paper explains how to animate 3D human model in OpenGL, using skeleton-driven deformation technique, similar to Linear Blend Skinning, that computes deformed weighted vertex position, which is calculated using quaternions. This helps to avoid some common animation problems, that occurs while dealing with various human body rigid parts transformations. The motion for this animation was captured using motion capture system and after processing it and converting to appropriate BVH format, was ready to use for animating 3D human model in lowest programming C++/OpenGL level.

Keywords: *Human animation, Vicon motion capture system, OpenGL, C++, Linear Blend Skinning*

1. Introduction

Our animation focuses on the most common and partly simple human body animation technique, that uses joints to animate human model. The joint structure, given their position and orientation, can be thought as being human body skeleton. The skin shape is associated to the joints, where as it's a 3D polygon mesh and it's the only thing that is displayed for the end-user. Due to very fast computation speeds, this technique is the most popular in animation production. On the other hand, using simple shape blending technique to deal with complex human body rigid parts transformations, there are various skin deformation problems. Typical ones are collapsing elbow, candy-wrapper joint when the arm turns 180 degrees, intersection between two adjacent bones (links) around a joint. Also such a technique don't consider many very complicated and detail human body deformations, for example dealing with muscles (stretch or bulge).

1.1. Previous works

What we've read and what was written there. References. [Linear blend skinning, for deformation problems, data formats, data format conversion problems (BVH) - this probably to be written somewhere else, maybe overview?]

1.2. Overview

What is represented in further sections?

TODO: Explain what are the problems, kinematic chain, what went wrong with stuff what we were doing, what we could tell to other groups in order to continue with project, emphasize that we use our joint model ant only take rotations.

2. Linear blend skinning

Linear blend skinning technique is widely used for interactive applications. It goes by many different names, such as Skeleton Subspace Deformation or SSD or "smooth skinning" in Maya.

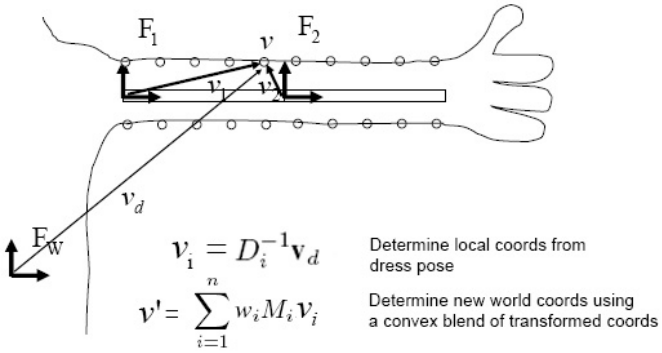
The linear blend skinning algorithm works by first placing a hierarchical skeleton inside a static model mesh of a character in some neutral pose (usually in the da Vinci posture or so called "dress pose"). Then, each vertex is assigned a set of influencing joints and a blending weight for each influence. Computing the deformation in some pose involves rigidly transforming each dress pose vertex by all of its influencing joints. Then the blending weights are used to combine these rigidly transformed positions.

The deformed vertex position, $\overline{\mathbf{v}}$ is

$$\overline{\mathbf{v}} = \sum_{i=1}^{n} w_i M_i D_i^{-1} \mathbf{v}_d$$

where w_i are the influence weights, v_d is the dress-pose location of a particular vertex $\mathbf{v}$, M_i is the transformation matrix associated with the *i*th influence, and D_i^{-1} is the inverse of the dress-pose matrix associated with the *i*th influence.

Taken together, $D_i^{-1}\mathbf{v}_d$ represents the location of $\mathbf{v}_d$ in the local coordinate frame of the *i*th influence.



Linear blend skinning though has two primary failings. First, the method is incapable of expressing complex deformations. Artifacts such as the "candy-wrapper", collapse effect on wrists and collapsing around bending joints as shown appear. They occur because vertices are transformed by linearly interpolated matrices. If the interpolated matrices are dissimilar as in a rotation of nearly 180 degrees, the interpolated transformation is degenerate, so the geometry must collapse. Second, authoring linear blend skins is difficult and frustrating for users.

Despite its failings, this skinning algorithm is very fast and widely supported by commercial applications so it remains popular especially in games and virtual environments.

3. Quaternions

A quaternion is a mathematical object, consisting of 4 scalars. Quaternions were first discovered in 19th century by Irish mathematician Sir William Rowan Hamilton during his search for a way to represent points in space. Soon after their introduction, quaternions appeared at most of universities as an advanced mathematics subject. Nowadays quaternions find their applications in signal processing, physics, bioinformatics, orbital mechanics as well as computer graphics.

In computer graphics Euler angles are known to have the disadvantage of being susceptible to "Gimbal lock" where attempts to rotate an object fail to appear as expected, due to the order in which the rotations are performed. Quaternions are a solution to this problem. Instead of rotating an object through a series of successive rotations, quaternions provide the ability to rotate an object through an arbitrary rotation axis and angle. Though the rotation is still performed by using matrix mathematics, instead of multiplying matrices together, quaternions, representing the axis of rotation, are multiplied together. The final resulting quaternion is then converted to the desired rotation matrix.

As mentioned earlier, a quaternion is four numbers, representing one real dimension and 3 imaginary dimensions. Each of it's imaginary dimensions has a unit value of square root of -1 (a complex number). In addition to that, imaginary dimensions are all perpendicular to each other and can be noted as i, j, k. So a quaternion can be represented as follows:

$$q = a + i*b + j*c + k*d$$

where a is a real dimension representation and b, c, d are just scalars.

Since quaternions are mathematical objects, they have a certain algebra with all the addition, subtraction, multiplication and division operations applied to them. In this section of the report we are going to consider just the quaternion multiplication operation, since this operation is necessary to represent a rotation. All other operations can be found in the references page of the report.

In order to perform a quaternion multiplication, a quaternion's imaginary dimension multiplication has to be described first:

$i * i = j * j = k * k = -1$

$i * j = k$

$j * i = -k$

$j * k = i$

$k * j = -i$

$k * i = j$

$i * k = -j$

The multiplication of two quaternions is :

$$(a + i*b + j*c + k*d) * (e + i*f + j*g + k*h) = a * e + i * a*f + j * a*g + k * a*h + i *b*e - b*f + k * b*g - j * b*h + j *e*c - k * c*f - c*g + i * c*h + k * e*d + j * d*f - i * d*g - d*h = (a*e - b*f - c*g - d*h) + i*(a*f + b*e + c*h - d*g) + j*(a*g - b*h + e*c + d*f) + k*(a*h + b*g - c*f + e*d)$$

The important thing to notice is that the multiplication of two quaternions is not commutative, so that

$$q1 * q2 != q2 * q1$$

As mentioned above, the multiplication operation, provides us with the ability to rotate one quaternion by another quaternion. So the rotation of quaternion q1 by quaternion q2 would result in another quaternion q = q2 * q1;

Since the representation of a quaternion is quite complex and hard to imagine, it can be interpreted in another way. The x, y, z components of a quaternion can be treated as a representation of rotation axis and a component - as a representation of rotation angle. The relation between actual values and their representations as quarternion's components can be described as:

- a = $cos(angle/2)$
- b = $axis_x * sin(angle/2)$
- c = $axis_y * sin(angle/2)$
- d = $axis_z * sin(angle/2)$

where $axis_x$, $axis_y$ and $axis_z$ is a normalized vector, representing the rotation axis.

In order to rotate a 3D point by a quaternion, a rotation matrix R has to be produced:

$$R = \begin{pmatrix} a^2 - b^2 - c^2 - d^2 & 2*b*c - 2*a*d & 2*b*d + 2*a*c & 0 \\ 2*b*c + 2*a*d & a^2 - b^2 + c^2 - d^2 & 2*c*d - 2*a*b & 0 \\ 2*b*d - 2*a*c & 2*c*d - 2*a*b & a^2 - b^2c^2 + d^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4. Motion capture

Motion capture (*mocap*) is sampling and recording motion of humans, animals, and inanimate objects as 3D data. The data can be used to study motion or to animate 3D computer models. During the motion capture process not only the capturing stage using *mocap* equipment is very important, as equally important are the preparation and post processing processes. The whole system must be well calibrated, adjusted and set up, furthermore, after capturing data needs to be cleaned, edited, and applied to a 3D model.

During this project, for capturing motion data, we used Vicon Motion System, combined with Vicon IQ 2.5 software running on our systems host pc. Details on how we managed to capture and process motion data can be found on the appendix **A**.

5. Motion data

There are various *motion data* storing formats, that mostly depend on what kind of program they are being used on. Our aim on this project, considering motion data, was to select the most appropriate data format. After reviewing some examples, the decision was made, that the best choice for our model animation would be using *BVH* motion data format. But because our motion capture is being done using Vicon IQ software, that's not able to directly export data to BVH file format, at first we had to work with Vicons' C3D and then convert it to BVH.

5.1. Description of Vicon C3D format

This section is meant to give just a brief introduction to the C3D file format, describing it in an abstract way. A reference to a detailed manual, describing the format, is given in the reference part of this document.

The C3D (Coordinate 3D) file format is a binary data file format originally developed for the AMASS photogrammetry software system, capable of storing 3D data as well as analog data together with all associated parameters for a single measurement trial. Since only the 3D data is of importance, we will not consider analog data in this document. The main advantage of C3D format over other motion capture data formats is that it is able to encapsulate the motion data as well as parameters, describing the motion data, in a single file. Apart from that, C3D is freely licensed and well documented.

Being a binary file, the C3D file consists of a number of 512-byte blocks. Logically C3D format can be divided into 3 basic sections, each having one or more 512-byte blocks:

- Header section is the first section of a file. The main purpose of this section is holding a pointer to the start of parameters section. Other parts of this section, is of no particular importance, as usually it consists data, copied from parameters section.
- The parameters section usually starts at block number 2, although this is not fixed and should not be assumed to be the case for every C3D file. This section contains information about the 3D data stored in the file. The section is extensible, meaning that user can define it's own parameters without violating the format specification.
- Data section containing the 3D point coordinates is usually located after the parameters section. This section simply contains sequential frames data. In the case of 3D points (the data is X, Y and Z coordinates).

5.2. Description of Biovision BVH format

The BVH format is an updated version of BioVisions BVA data format, with the addition of a hierarchical data structure representing the bones of the skeleton. The BVH file is an ASCII file and consists of two parts.

First section (*hierarchy*) is for storing hierarchy and initial pose of the skeleton, basically joint-to-joint connections and offsets. While as second section (*motion*) describes the channel motion data for each frame, that is describes the movement of individual joints.

The *hierarchy* section starts with *root* joint and contains the definition of a joint hierarchy within nested braces like source code written in the C programming language. Each joint in a *hierarchy* has an *offset* field and a *channels* field. The *offset* field stores initial *offset* values for each joint with respect to its parent joint. *channels* field defines which channels of transformation (translation and/or rotation) exist for the joint in the *motion* data section of the file. The *channels* field also defines the order of transformation. A

channel is either x-, y-, or z-translation or local x-, y-, or z-rotation. All the segments are assumed to be rigid and scaling is not available. The *end site* field is also available in order to determine body segment end.

In the *motion* section the total number of frames in the animation and the frame speed in frames-per-second is defined. Every next row then contains data values for all *channels* which were specified in the *hierarchy* section. The listing order of *motion* values in each row in is assumed to match their listed order from the *hierarchy* section (top down).

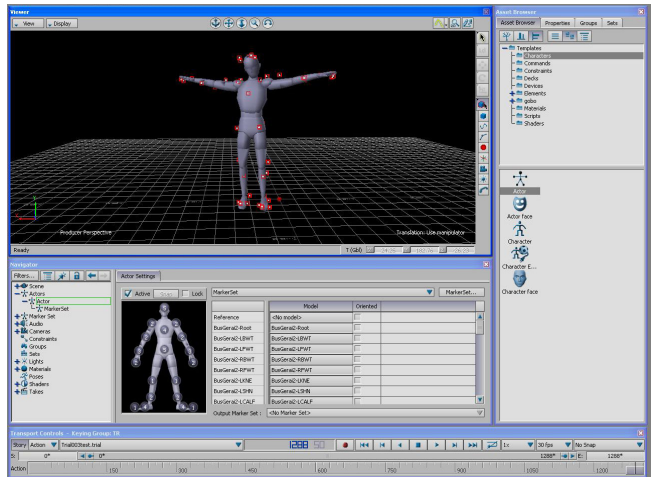
There are few drawbacks of the BVH format. One is that it lacks a full definition of the initial pose. Another drawback is that the format has only translational offsets of children segments from their parents. No rotational offset can be defined. Moreover, the BVH format is often implemented differently in different applications, that is, one BVH format that works well in one application may not be interpreted in another. All the same the format is very flexible and it is relatively easy to edit BVH files.

[Give more details later about exact structure of our model (like example)]

5.3. Processing motion data

During the workflow of our project, the motion capture data, produced by the Vicon motion capturing system using Vicon IQ software, was encoded in C3D file format. This data in C3D file contains only the positions of the reflective markers at each frame (the same positions that were used in Vicon IQ, animating labeled skeleton). As it is not sufficient for us to correctly animate character, we have to do additional data processing.

We import C3D animation data file to MotionBuilder software. Then the whole data processing is similar to the one, that must be done after capturing data in post-processing stage, using Vicon IQ software. We assign the C3D motion data to so called actors and characters in MotionBuilder. Basically actor here is a mesh models, and character is skeleton with specified joint structure. As characters have their own skeleton, after assigning C3D data and using the inverse kinematics technique, we have whole body hierarchy and can export to BVH format, which contains the rotation and translation data of each joint for each frame.

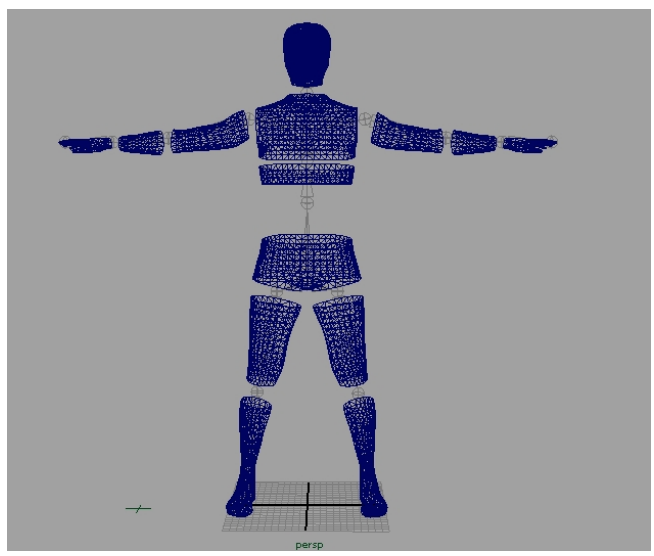


6. Human body mesh model

What do we use for our animation?

6.1. Mesh model preparations in Maya

What Mesh model in Maya do we use? How do we prepare it, that it would be suitable for out program? How we cut mesh to different body parts, export to .obj format files.



6.2. Mesh model file format used in animation

Why do we use .obj file format? How does it look like?

6.3. Description of OBJ format

OBJ is a geometry definition file format, first developed by Wavefront technologies. The file format is open and has

been adopted by most of 3D graphics application vendors. It can be imported/exported from most of 3D modeling tools such as Autodesk's Maya, 3ds Max, Newtek's Lightwave, Blender, etc. This section of the document provides a thorough explanation of the most important parts of an object file, with more details provided in the reference page.

An object file can be stored in ASCII (using the ".obj" file extension) or in binary format (using the .MOD extension). The binary format is proprietary and undocumented, so only the ASCII format is described here.

The OBJ file format supports lines, polygons, and free-form curves and surfaces. Lines and polygons are described in terms of their points, while curves and surfaces are defined with control points and other information depending on the type of curve. Since, in our project, lines and polygons were sufficient to represent the model appropriately, only the parameters, concerning, these entities are described below.

The OBJ file is composed of lines of text, each of them starting with a token, which describes the type of the entity being recorded in that line. Below are listed various tokens, which were of importance in our project:

- "#" - a comment line. Lines, starting with "#" token, are simply skipped by OBJ file readers. For example "# this is a comment".

- "g" - a group line. Lines, starting with "g" (group) token, determine the start of a group. "g" token is followed by the group name. For example "$gleft_arm$". In our project each group name corresponded to some body part, described by vertices, textures, normals and polygons.

- "v" - a vertex line. Lines, starting with "v" (vertex) token, provide the information, concerning vertices. This token is followed by x, y and z coordinates of the vertex. For example "v -0.756447 0.702621 0.047024" is a vertex with coordinates (-0.756447, 0.702621, 0.047024).

- "vt" - a texture line. Lines, starting with "vt" (vertex texture) token, are recorded with information, concerning textures. "vt" token is followed by x, y and z coordinates of the texture, although only the x and y coordinates are of importance (z coordinate is 0.0). For example "vt 0.487840 0.942165 0.000000".

- "vn" - a vertex normal line. Lines, starting with "vn" (vertex normal) token, contain information, concerning the normal of a vertex. "vn" token is followed by x, y and z coordinates of the normal. For example "vn 0.149280 -0.186998 -0.240511".

- "f" - a line, describing face. Lines, starting with "f" (face) token, provide the information, concerning polygons. "f" token is followed by a number of triplets, which is equal to the number of vertices, the polygon has. Each triplet is of a form "int/int/int", where the first "int" is a vertex position in the file, the second "int" is "texture" position in the file and the third "int" is a normal position in the file. For example "f 6/9/6 2/5/2 1/1/1" is a triangle polygon, with vertices 6, 2 and 1, textures 9, 5 and 1 and normals 6, 2 and 1.

The above described tokens were of importance in our project, but they are not the only ones that OBJ format allows. The OBJ format specification is much broader and covers such abilities as surface encoding, connectivity between free-form surfaces, rendering attributes. The OBJ specification can be found in the document's reference page.

7. Animating human body

Explain how we load human model, exported from Maya to .obj file. How we create natural primary human pose, assign meshes to joints and so on.

7.1. Loading human body model in OpenGL

How we load human body to OpenGL. How we import mesh from obj files. Joint creation. Joint connection with meshes.

7.2. Linear blend skinning relations

How we adapted linear blend skinning to our animations. Explain how our method using meshes assigned with joint rotations is similar to linear blend skinning. How we automated all the things.

8. Conclusion

Conclusion

8.1. Future work

What could be implemented to improve this project? Live streaming to our animating program (straight from cameras to animation in OpenGL); other skin deformations

A Details of using Vicon motion system

A.1. Vicon MX motion capture system

Vicon MX motion capture system is a passive optical system that uses markers coated with a retroreflective material to reflect light back that is generated near the cameras lens. The camera's threshold can be adjusted so only the bright reflective markers will be sampled, ignoring skin and fabric.

An object with markers attached at known positions is used to calibrate the cameras and obtain their positions and the lens distortion of each camera is measured. Providing two calibrated cameras see a marker, a 3 dimensional fix can be obtained.

Our system used for this project mainly consists of:

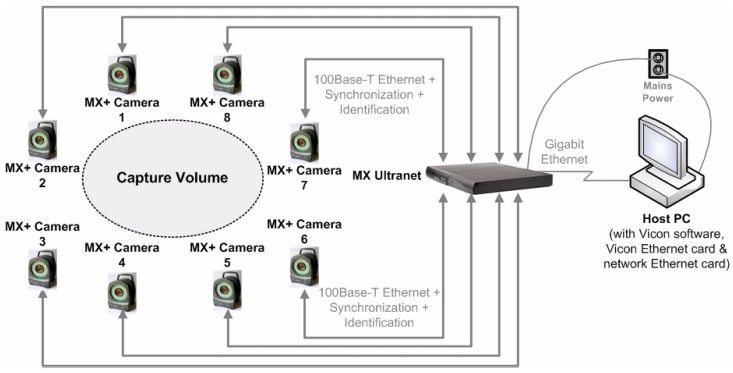
- 8 Vicon MX3+ cameras, fitted with sensitive solid-state sensors, with stringent checks for linearity, sensitivity, and absence of jitter. Each MX camera is programmed with firmware to control its operation and enable it to perform its own onboard grayscale processing. Cameras and their relevant characteristics are recognized immediately when they are plugged in to Vicon MX system.

- MX Ultranet, that supplies power, synchronization, and communications for all eight connected MX cameras. Only MX Ultranet is connected to the Host PC and routes all communication to and from the host PC, and timing/synchronization signals to and from the MX cameras connected to it.

- Host PC, with seperate Ethernet port (preferably Gigabit ethernet card) to enable communications between the Vicon software installed on this *host pc* (in our case *Vicon IQ 2.5*) and *MX Ultranet*. No other special requirements for Host PC, but processor and memory are vital for being able to achieve better results. Our *host pc* had 2.49GB of RAM and 1.60GHz Intel Xeon 8 core processor. *Vicon IQ* software on a *host pc* is used to collect and process the raw video data. It takes the two-dimensional data from each camera, combining it with calibration data to reconstruct the equivalent digital motion in three dimensions. This can be viewed as a virtual three-dimensional motion. After this reconstruction the data may be passed to other applications.

A.2. Motion system preparations

Vicon motion capture system high-resolution cameras are situated in a circle, above the maximum height of any



marker that would be captured, that forms a motion capture space. Then the capture volume is created. The capture volume is three-dimensional, but in practice is more readily viewed by its boundaries marked out on the floor. This gives you a guide for the positioning of the cameras, and your subjects a guide as to where they can move. The area should be as central to your capture space as possible to maximize the volume you are able to capture. Initially you are looking to see whether the camera is positioned and oriented for the maximum view. Then every single camera can be adjusted separately by changing software settings, to achieve best result.

After setting up cameras, the vital thing is system calibration. It allows the software to calculate the relative location and orientation of all the cameras. When your movements have been recorded the reconstruction process uses these measurements to calculate the accurate movements of the markers through space.

This is done by static and dynamic calibrations:

- Static calibration - calculates the origin or center of the capture volume and determines the orientation of the 3D Workspace.

- Dynamic calibration - involves movement/waving of a calibration wand throughout the whole volume and allows the system to calculate the relative positions and orientations of the cameras. It also linearizes the cameras.

A.3. Capturing motion data

Having the whole motion system ready for use, the next first step is placing all the retroreflective markers on special suit. The special suit is used to achieve the most accurate results in motion, that motion system gets by tracking those reflective markers and it should be as as tight as possible. Though the suit is not always used while capturing motion data, as for example capturing face motion data, special reflective markers are used, that are put directly on subjects

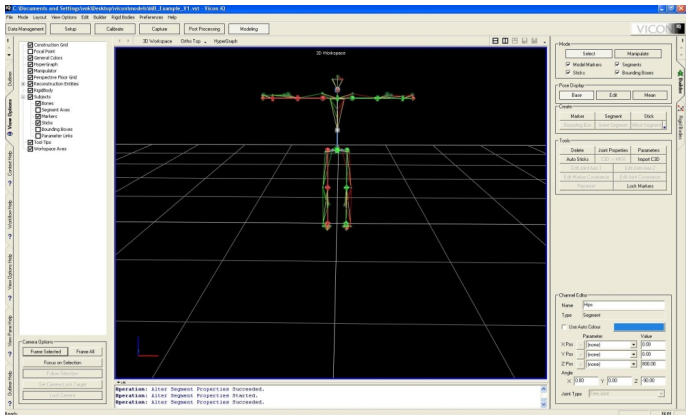
face. Placing markers is vital that the animating template is already selected. In this case, skeleton template, that will be assigned with motion data points. These templates describes the bone structure, joint positions and markers positions, how these markers relate to the bone structure and joints.

When markers are ready, it's time to start capturing data. Motion capture subject should stand in the middle of the circle in the capture area, with his back straight, arms straight out with palms facing down, and feet forwards. This is called a T-pose, and every capture session should begin and end every in it. This pose is common in animating human bodies, as it allows meshes of the skin to be connected correctly. Also it usually allows the motion capture system to determine all the markers positions right from the frame 1, as all of them are in its visibility area. While capturing, you should be aware, that during whole capture process, you should be in cameras capture area.

A.4. Captured data post-processing

After capturing motion data, in order to proceed further, post-processing must be done. Firstly it involves labeling/matching each marker from captured data to our used animation template (skeleton template) at the first frame. Only by labeling these markers, we are getting logical motion data in further frames, and not some meaningless points in the 3D world. This is the stage, when it's necessary that those reflective markers were situated correctly on the motion capture subject.

Though labeling does really big part in getting correct motion data, it's very important to do some manual tweaking of the labeled points and track unlabeled ones. They can be unlabeled either of some motion capture failure at some point or there can be a "ghost" points that needs to be deleted. Ghost points are quite common while using passive optical motion capture system with retroreflective markers.



Without manual fixing, special pipelines are used to automate various processes, for example using kinematic fitting to match our animation template to our labeled markers. Various pipelines can be used to significantly shorten the post-processing time.

References

[1] F. S. Hill and S. M.Kelley. *Computer Graphics Using OpenGL*.

[2] M. Kitagawa and B. Windsor. *MoCap for Artists - Workflow and Techniques for Motion Capture*.

[3] M. Meredith and S.Maddock. Motion capture file formats explained. 2003.

[4] A. Mohr and M. Gleicher. Building efficient, accurate character skins from example. 2003.

[5] A. Mohr, L. Tokheim, and M. Gleicher. Direct manpulation of interactive character skins. 2003.

[6] Motion Lab Systems, http://www.c3d.org/HTML/default.htm. *C3D Reference*.

[7] Vicon Motion Systems Ltd. *Vicon system reference*.