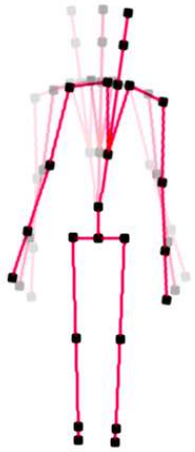
Design and implementation of real-time character animation library

Håkan Almer & Erik Erlandson

August 1, 2005



Abstract

This report describes the design and implemention of Open Skeleton Library, a character animation library for realistic looking human based characters with a skeleton structure. The library includes most elements found in other modern character animation libraries. Additionally new methods of control and animation to make individual characteristics of characters easier to achieve are included. A technique to import animations from one model format to another by the use of skeleton matching is also described.

Contents

1 Introduction

The increasing demand for games and other applications that use detailed character animation in varied situations, such as a large crowd simulation or a close view of a human speaking, creates a need for more advanced character animation libraries. Almost all new games use some sort of character animation and very often advanced techniques, for instance, motion capture for data acquisition. Flexible ways to handle different methods and integrate them is required to avoid spending heavy resources on developing new modelling file formats. Most commercial game engines such as *Cipher* and *Unreal Engine 3* have a very extensive and feature rich support for character animation. There are also open source systems which are quite extensive, but the commercial systems are superior in most aspects (see Figure 1 and 2). However, most of the systems, are integrated in a game engine, which makes them inaccessable outside that system.

As the goal of this project a new library superior to any other stand-alone library was designed and implemented. It includes modern features and new methods for animation, especially for easily variable animations and adaptation between modelling formats. These varied animations are well suited to use, for instance, on large crowds to simulate individuality while still using just a few models. This increases the credibility of a scene while not imposing much custom programming on the creator. The adaptation, or model/skeleton fitting, was primarily designed to animate any skeleton or model using a set of motion capture data from another source. The library was designed for modularity and useability, which makes it suitable for integration in game engines or for stand-alone applications using arbitrary modelling file formats.

Feature	Cal3d	Irrlich	Nebula2[a]	OSL[b]
Skeleton animation	X	X	X	X
Mesh animation		X		
Blending	X	X	X	X
Weighted skinning	X		X	X
Level of detail	X	X	X	
Vertex shader skinning			X	
Inverse kinematics				
Skeleton fitting				X
Function controlled effects				X
Rigid bone attachment	X	X	X	X
Stand-alone	X			X

[a]Nebula Device 2
[b]Open Skeleton Library

Table 1: Open system's features

1.1 Outline of Thesis

The first part of this report deals with the theory of character animation, both established theory and new additions developed for Open Skeleton

Feature	Cipher	T3D[a]	TV3D[b]	UE3[c]
Skeleton animation	X	X	X	X
Mesh animation	X	X	X	
Blending	X	X	X	X
Weighted skinning	X	X	X	X
Level of detail	X	X	X	X
Vertex shader skinning	X		X	X
Inverse kinematics				X
Skeleton fitting				
Function controlled effects				X[d]
Rigid bone attachment	X		X	X
Stand-alone				

[a]Torque 3D
[b]True Vision 3D
[c]Unreal Engine 3
[d]Procedural skeletal controllers

Table 2: Commercial game engine's features

Library. Next, the report presents a review of character animation in some of the leading game engines and animation libraries, both commercial and open source, which is found in Section 3. The goal of this review was to determine what the existing systems lack and how to build a better system. In Section 4, the design of Open Skeleton Library will be presented. This part covers how OSL deals with the theory found in Section 2. The next section show how easy the library can be used, which is illustrated by some example applications. Finally, results and conclusions are drawn from what this report covers.

2 Theory and Concepts

In this chapter the theoretical issues of classical character animation and skeleton structures are described. Additionally new concepts developed for OSL are covered.

2.1 Keyframe Animation

Storing every step of an animation would require a lot of memory. It is also hard to define what a *step* is as the model would move different distances between screen-updates depending on the speed of computer used. A solution to both these problem is a method of animation called *keyframe animation*. In this method the stored keyframes are further apart and are interpolated between to get the position of the model at a specific time. By only storing relatively few keyframes there is a large reduction in the memory needed. Depending on how close the keyframes are will determine how precise the animation will be, but this comes with the cost of extra memory used to store the keyframes. This method is used with both Mesh animation and Skeleton animation, which will be described in the next two sections.

Figure 1 shows a part of an animation sequence with three keyframes k0, k1 and k2. The time t shows the current position of the animaiton and the next position $t + \Delta$ to be sampled. It will then simply use keyframe k1 weighted by a factor 0.4 and k2 by 0.6 producing an appropriate mix between the two. Figure 2 shows an example of three sequential keyframes from a ballet animation.

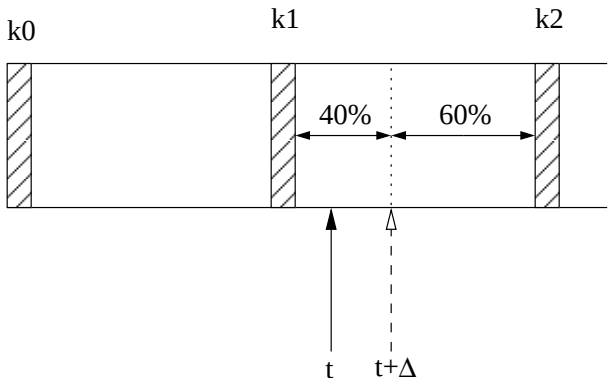


Figure 1: Sampling of keyframes

2.2 Mesh Animation

Mesh animation or *vertex animation* is perhaps the simplest method of character animation. It works by building keyframes in a modelling software which linear interpolation are then applied to. A keyframe consists of a complete mesh of the character at a particular time in the animation. Mesh animation suffers from several severe drawbacks. When performing linear interpolating between two keyframes the relative dimensions of the polygon will not always be maintained [1]. A wrongly interpolated

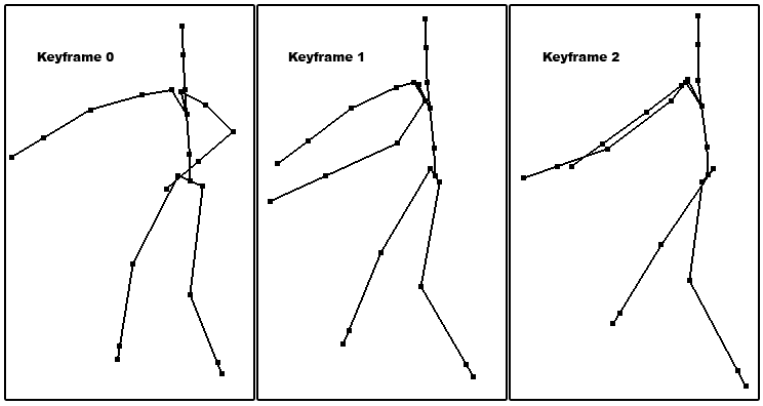


Figure 2: Ballet animation keyframe

point is shown in Figure 3 where the indented arch is not followed and the dimensions are changed, which result in deformation of the mesh. This problem only gets worse when the animations are blended. Deformation can be limited by starting with more keyframes. But this only makes the other large problem mesh animation has more severe. For each keyframe the position of every vertex needs to be stored. For complex characters this create a high memory demand and the increase in keyframes will result an increase in memory used. Additionally all extra keyframes need to be modelled, which is time consuming. These limitations is why most modern game-designers no longer use mesh animation.

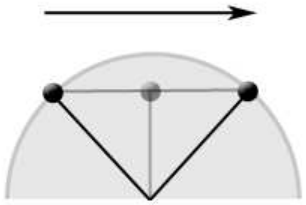


Figure 3: Mesh deformation

2.3 Skeleton Animation

The main reason for introducing a skeleton to animate characters is to overcome some of the problems that arise from simple mesh animation. As described in Section 2.2, serious problems arise during keyframe interpolation but there is also the problem of having to animate every different models mesh. By using a skeleton as an underlying structure, the mesh can be attached to it, following the skeletons movements in an appropriate way. This corrects the problem that arise with mesh animation. Figure 4 illustrates the indented result of Figure 3, following the arch correctly. How to make the mesh follow the bones are described in the next section.

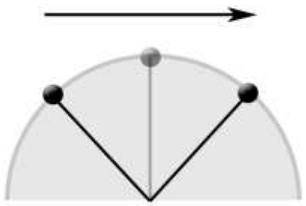


Figure 4: Correct interpolation

The most basic skeleton is built by a hierachical structure of joints which contain a 3x3 rotation matrix or quaternion and the length of the bone, often given by the offset to its parent in the hierachical structure. A simple setup of a skeleton structure can be achieved with:

```
Matrix jointRotations[nbrOfBones]
Vector jointOffset[nbrOfBones]
int parentBone[nbrOfBones]
```

Using a setup of this type, a rotation matrix and offset for each joint, the joints global location, J_k^G, can be transformed to their new correct position using the following recursive calculations. The result of this can be seen in Figure 5.

$$\begin{aligned} \Theta_0^G &= \Theta_0^L \\ \Theta_k^G &= \Theta_k^L \cdot \Theta_{k-1}^G \\ J_0^G &= J_0^L \\ J_k^G &= J_{k-1}^G + \Theta_k^G \cdot J_k^L \end{aligned}$$

G and L indicate global model space[1] and local bone space respectively. J is the joint offset to the root node in global model space and offset to its parent in bone/joint space. The k:th joint's global rotation matrix is Θ_k^G. Incidently, this joint propagation technique is also known as *forward kinematics*.

2.4 Skinning

Skinning is the technique used for *sewing* a mesh onto the skeleton. The main idea is to attach a special mesh to the skeleton and have it follow it as it moves. A straightforward way of doing this is by assigning every vertex of the mesh to a joint. The vertex is also given an offset to the joint, which is the vertex's position relative to the joint. The mesh's vertices are then transformed by the bones rotation and thus fitted over

[1] The different spaces will refer to the different transformations used. More on this can be found in most modern computer graphic books.

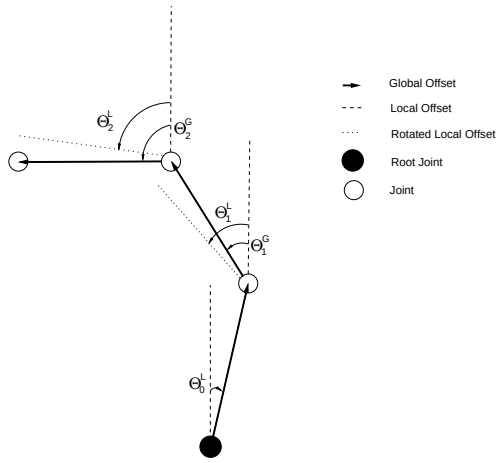


Figure 5: The skeleton structure with rotations

the bone. This simple scheme gives a disturbing artifact because of the unability to deform the submeshes naturally, as the human body would do. The desirable effect would be that they join together with each other by the joints smooth manner. There exist a solution for this problem but it comes with the price of complexity both for the mesh and algorithms. The solution is based on the ability for a submesh's vertex to hold different weights linking it to several bones. All of these bone's transform's will influence the final position of the vertex depending on their weight resulting in the following transform equation for a single vertex V:

$$V^G = \sum_i (J_k^G + \Theta_k^G \cdot W_i) \cdot w_i \ , \quad \forall \ i \tag{1}$$

, where k is the joint that the weight is attached to, W_i the offset of weight i and w_i the influence that the weight has on the vertex. The weights for each vertex should fulfill the criteria $\sum_i w_i = 1$. Normally no more than four weights are used or neeeded for each vertex. To transform the vertices' normals the following, similar transform is used[2]:

$$N^G = \sum_i (\Theta_k^G)^{-1^T} \cdot N_i^L \cdot w_i \ , \quad \forall \ i \tag{2}$$

2.5 Motion Capture Techniques

Animating realistic motions, especially for humans, is very difficult. The slightest deviation from a motion in real world is usually instantly spotted by any human observer since we are trained from birth in what should be the correct motion. Instead of trying to *guess* the correct motion the use of motion capture techniques are introduced. The common procedure to record the movement data is for a human actor to wear reflective data markers and be photographed by special cameras, sampling the space positions with a specified frequency. This data is then transformed from

[2]Calculating the transpose of the inverse is not necessary when the matrix is orthogonal, which is the case for rotation matrices, since $M^{-1} = M^T$ and $(M^T)^T = M$ holds.

position to rotation of the limb for which the marker belongs to. The process is neither easy nor cheap but in some cases necessary for the realism it brings. There exists other ways to obtain the data, for example by the use of special sensors at the desired points registering the movements from the actor mechanically.

2.6 Blending

Often it is not feasible to make new animations for every motion a character will have to go through. Making one animation for walking and one for running is no problem, but one would like to have all the intermediate states in between the two to accomplish a smooth transition. It would not be economical to make new animations for each combination which is why the *blending* method is introduced. It works similar to keyframe animation in that it interpolates between two keyframes. However, instead of two keyframes of the same animation, seperated in time, it interpolates between two keyframes of different animations. Using this it is enough to have only one animation for walk and one for run, and then blend between the two to get any intermediate state. The process is simple, but can save a lot of modelling time and increase flexibility for a relativly small increase in processing.

This is all good for full body animation but a problem arises when only the influence of certain bones are used for the action. Imagine the common scenario where the character is walking and the user triggers an action, for instance the actor waving with one hand. If just a simple blend, using equal parts from each animaiton, is used the actor will wave with the arm and hand but the arm animation from the walk will have a 50% influence on the wave animation resulting in a half wave. Also the walking animation will be disrupted by the wave animations all other inactive bones. This problem can be alleviated by keeping extra information for all bones in an animation to flag if its active of not. Model formats such as MD5 used in the game *Doom 3*[3] stores 6 bits of information for each bone of which components it holds, where the components are X, Y and Z rotations and offsets[4]. This scheme enables a triggered animation to have priority over the bones that it influences and replacing them with the triggered animation. Note that this method replaces the blender with a simple on/off check for each bone or component. For model formats that do not store this information it can be obtained at runtime by the blender or by processing the keyframes offline.

2.7 Inverse Kinematics

Calculating the end point from a series of joint rotations is called forward kinematics. In *inverse kinematics* the end point is known and the goal is to calculate the joint rotations. This is a much more complex problem because of several factors. For instance, the problem does not always have a single solution, even when dealing with few bones. Also there has to be some constraints on the joint rotations or else the joints may bend in a completely unrealistic fashion. To stop this; some kind of joint constraints are usually implemented, generally by assigning degrees of freedom, or DOFs, to the joints as well as placing limits on the rotation angles. Other

[3] `http://www.doom3.com`
[4] this is used to vary the size of the bones

ways to achieve this include measures such as collison detection to avoid moving a part of the model through another. Further, there is often some schemes that tries to limit how far the bones stray from a desired configuration or minimize how much each joint rotates. Since it would be impractical to use inverse kinematics on a whole skeleton, which often has 50 or more DOFs, inverse kinematics often divides the skeleton in several branches. When animating the skeleton, the algorithm will only calculate as far back as the branch root node. These branch root nodes are usually the shoulder and hip joints with, for instance head movements calculated back to the original root node.

2.8 Vertex Shader Driven Skinning

One of the primary bottlenecks in skeleton based character animation is the transformation of the mesh, the skinning. This can be alleviated by the use of vertex shader driven skinning.

With a programmable graphics card it is possible to modify the output of what is sent to the GPU[5]. Modern cards can be set to operate per vertex and/or per pixel which has been given the names *vertex shader* and *pixel shader*[6]. Special programs, called shader programs, are loaded into the graphic cards memory and subsequently the vertices are sent there for the code to handle. Popular shading languages are *n*Vidia's *CG*[7], Microsoft's *HLSL* and the extension of OPENGL; *GLSL*. The reason for using shaders is the possibility to do nice effects in real-time, such as fur, animated bump mapping and a lot more which was previously not possible to accomplish at run-time. The GPU being designed to process graphics does heavy computations such as matrix multiplication in a fast way, which is why it is desirable to move the skinning computations to this processor.

Recalling Equation 1, the same translation of vertices is wanted for this technique but is applied to the vertices by sending it to the GPU in their original form and then transforming them by the vertex shader at the GPU [2].

2.9 New Concepts

This section describes new concepts implemented in Open Skeleton Library. These techniques are not found in other libraries. However, less advanced variants of the function controlled effects described in this section can be found in modern commercial systems, such as *Unreal Engine 3*.

2.9.1 Skeleton Fitting

The need for *skeleton fitting* arises when an animation for a certain model is desired to fit another skeleton. This means using the same animation of bones for a different skeleton model with a different configuration of bones. A function is therefore sought to transform the animation keyframes for the original model A to another model B. For each keyframe the new joint rotations $\widehat{B_{i,k}}$ will be calculated using A:s joint rotations.

[5] Graphics Processing Unit
[6] Also sometimes refered to as the fragment shader.
[7] www.nvidia.com

The simple approach to accomplish this is to locate the corresponding bones in the two skeleton models. Furthermore, if the file format use a naming or indexing scheme for the bones that are the same for every model, it is possible to create a generic function that performs the fitting between these models. One still has to carefully oversee that the bind pose corresponds between the two formats. This is normally not the case and therefore a slighty more advanced method must be applied which is described below.

Mapping with Offset Compensation

To illustrate this method two example skeletons are used, depicted in Figure 6. It is obvious that these two skeleton can not be matched exactly. One method is to make a map of corresponding bones that should rotate to create a similiar motion. They are easily identified by hand and it is not a very challanging task for human. For instance looking more closely on the models left shoulders as depicted in Figure 7 the following bones match:

- A:Chest - B:Chest
- A:Shoulders - B:Chest2
- A:LeftShoulder - B:LeftShoulder
- A:LeftElbow - B:LeftLowerArm
- A:LeftWrist - B:LeftWrist

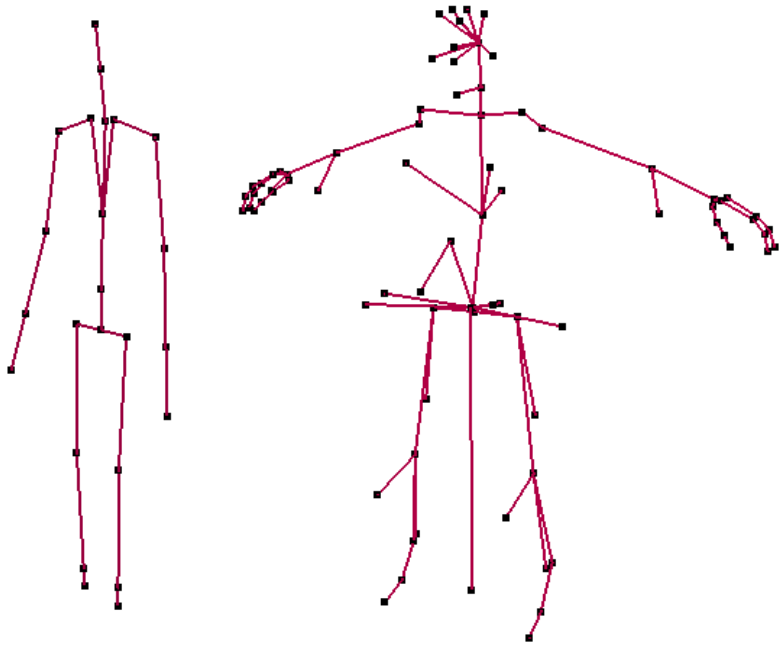


Figure 6: Two skeletons with different skeletons

Note that *A:LeftCollar* and *B:LeftUpperArm* has been left out since no apparent match exists for them. Another possible scheme would be to leave *B:LeftShoulder* out and use the pair *A:LeftShoulder* and *B:LeftUpperArm*

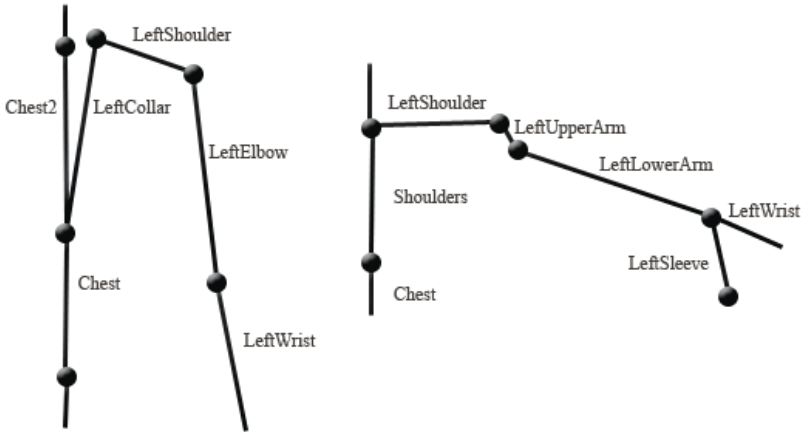


Figure 7: Close up of left shoulder and arm

to avoid the low shoulders model A would get. Moving on, the problem with the different bind poses of the models is addressed. Using the suggested mapping above, three of the matching bone pairs are shown in Figure 8. The interest here lies in their offset. This defines the bones' directional vector which in turn gives the pair's offset angle ϕ, which is the desired compensation for this particular bone. A difference or compensation rotation Φ_i can then be derived by defining a rotation ϕ degrees around the axis given by the cross product $V_A \times V_B$, where V is the directional vector of the bone.

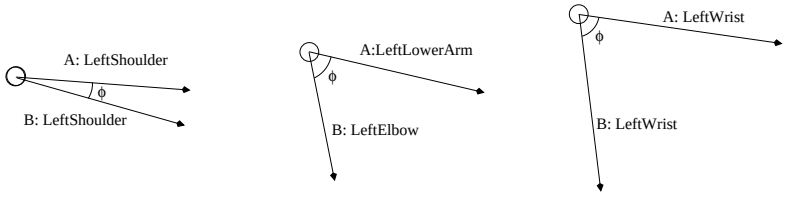


Figure 8: The matched bones bones and their angles

Since its common to define the bones rotation at its parent bone (see Figure 12), that is, building the skeleton with joints, there exists no possibility to rotate the bones correctly by using only the animation's rotations for joints that has more than one matched child. To alliviate this the bind pose must be adjusted by setting new offsets for the joints. However, using this solution may cause problems with the mesh since it also must be compensated.

Other Possible Techniques

The next approach of doing a skeleton fitting would be of a more analytical nature. The ultimate goal would be to build a new animation sequence given any two skeleton model formats and an animation belonging to one of them. The additional challenge to this method compared to that of the manually mapped offset compensation is to produce the bone mapping in

an algorithmical manner. Possible ways to achieve this would typically involve exploiting the knowledge of the proportions of the human body, at least when dealing with humanoid forms, to derive what bones should be matched.

2.9.2 Function Control Effects

Given a motion pattern connected to a set of skeleton bones that is beautifully animated, it could be disirable to change this pattern slightly to give a wide variety of flavours. This is achieved by the use of *function controlled effects* that is attached to different parts of the character depending on its definition.

The possibilities of this technique are numerous. For instance it can be used on a crowd of people performing the same animation, but with slightly different effects, producing individual smooth and elegant motions without being forced to create a unique pattern for each character. Further uses could include an implementation of inverse kinematics or parametric dynamics[8] as described by Ciechomski[3].

By attaching an effect as a function to a bone, the entire skeleton or even a model's mesh can produce a controlled deviation which will alter a model in a flexible way. Some examples of simple skeleton effects would include:

- A slight forward bent applied to the backbone to give the character a slightly tired appearance
- Letting the spine have a bone oscillate slowly to simulate drunkeness
- Adjusting the rotation of one leg to make the illusion of a crippled leg

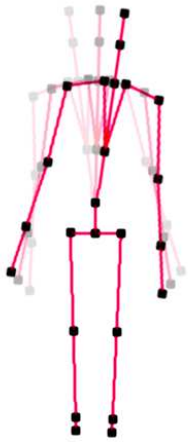


Figure 9: Skeleton with oscillating backbone

One advantage of using effects for creating differences in a crowd of people is that it can reduce the calculations performed for each model. This is done by doing all skeleton and mesh calculations for the entire model and

[8] A very quick animation, such as the reaction of being shot.

then applying some effect that would affect some bone in the skeleton to change its rotation. Note that new locations for this bone and its children would need to be recalculated, nevertheless this approach would cut the calculations for those bones not affected by the effect since the recalculation of an entire model would not be necessary.

3 Previous Work

This chapter deals with existing systems that implement some form of character animation. A brief description, as well as a list of features, is provided for each system. Finally, comments on the systems abilities are presented.

3.1 Open Systems

The selection of open systems was made by first performing a thorough search, finding every library with character animiation support, and then select some appropriate systems. *Cal3d* was choosen because it was the only stand-alone library found. *Nebula Device 2* and *Irrlich* because of their popularity in the open source game engine community.

3.1.1 Cal3d

Cal3d is an open source character animation library which is included or available as a plugin to some open source game engines like *Crystal Space*. To quote the projects homepage:

> *"Cal3d is a skeletal based 3d character animation library written in C++ in a platform-/graphic API-independent way."*

This is the only existing open source character library not belonging to a certain engine which meets an acceptable standard. It is a nice system that can be added to a graphic engine that lacks character support. All basic concepts of character animation exist in this library but some advanced features, such as weight controlled meshes, are not available. On the other hand, its ability to blend animation is very good. It uses a native model file format and runs other formats by first converting them with different exporters.

Features

- Skeleton based animation
- Powerful animation blending
- Automatic level-of-detail
- Fully controllable characters
- Exporter for 3D Studio Max and Milkshape 3D
- Stand-alone library

3.1.2 Irrlicht

The Irrlicht engine is a rather fast game engine with support for both mesh and skeleton based animation. This is an appropriate engine for applications with basic character animations, but unable to perform advanced effects. The main focus is clearly not on character animation which leaves much to be desired in the area of advanced character features and control (see introduction section Table 1).

Features

- Character animation system with skeletal and morph target animation
- Support for blending and level-of-detail

3.1.3 Nebula Device 2

Nebula Device 2 is an advanced graphics engine with support for most modern character animation features. It uses a palette-skinning vertex shader to perform skinning, which the webpage claims it is the most effective way to do skinning on a Win32 machine. Vertex shader driven skinning permits large improvements in performance and also ease the workload for the processing unit. Since the engine has a good character animation system and a high performance it is, in the opinion of this report's authors, the best open source game engine available especially concerning character animation.

Features

- Animation data optionally completely in memory, or streamed
- Streaming useful for large cutscenes
- Animation data compression
- Any type of 4-dimensional data, not just translation, rotation, scaling
- Rotation animation through quaternions, not Euler angles
- Weighted animation blending
- Vertex skinning with 4 weights per vertex
- Any number of bones per character
- Any number of skin meshes per character
- Skinning running completely in the vertex shader : no per-vertex operations on the CPU

3.2 Commercial Systems

There are many games today that implement advanced character animation systems. These engines are typically built for speed and have their own modelling formats and structure. This section presents some of these systems.

3.2.1 Cipher

The Cipher webpage states:

> *"Cipher's Advanced Animation System offers powerful features and integration with Character Studio and Biped in 3ds max. The characters in your game can be animated with Cipher's sophisticated skeletal animation system, while simpler objects can take advantage of the flexibility of the vertex animation system"*

After examining the feature list and evaluating this engine the impression is very positive. It has many advanced features beyond the basics. Such features include detailed animation possibilities for facial features and hair and full control of the skeleton, enabling extensions easily. The licence comes with exporter tools, which enables the use of several model file formats.

Features

- Arbitrary number of bones per character.
- Arbitrary bone influences per vertex.
- Deformable skins with multiple levels of details.
- Animate facial features and hair.
- Attach rigid models to individual bones.
- Full access to animating bone information (e.g. to place gun in characters hand).
- Vertex animation supports arbitrary deformation of models.
- Smooth blending between frames in both skeletal and vertex animations.
- Variable speed and reverse playback of animations.
- Efficient use of memory and extremely fast skinning.

Licence fee: $100
`http://www.cipherengine.com`

3.2.2 Torque 3D

The feature list for *Torque 3D* reveals insufficient animation library. Presenting trivial features like *multi-bone* skeleton, mesh and texture coordinates, which are considered intrinsic properties of any character animation system, is a sign of thin character animation support. However, a scripting feature is a convinient addition to any library.

Features

- Animation: multi-bone skeletons, mesh, texture bitmap, texture coordinates and visibility
- Mesh vertex deformation animation
- Real-time animation blending for dynamic, flexible character actions
- Scripting interface to multi-sequence animation manager
- Projected object shadows (clipped against the environment)
- Level of detail support

Licence fee: $100
`http://www.garagegames.com/`

3.2.3 True Vision 3D

True Vision 3D is a basic game engine that offers nothing more than what is available in any modern game engine, except for the use of rigid bone attachments and custom bone rotations. The focus is not on character animation but rather the sound and media library.

Features

- Powerful animation system
- Skeleton-based, Keyframe-based, or Morph-based animations
- Attach child meshes to bones
- Animate via custom bone rotations

Licence fee: $150
`http://www.truevision3d.com`

3.2.4 Unreal Engine 3

Unreal Engine 3 is one of the best game engines available to date, but only available for those who can afford it. The feature list it presents does not bother to cover the basics and instead displays a list of impressive goodies. According to the official presentation of the engine, it features a skeleton animation system combining motion capture animation with physic feedback and procedual controllers[9]. A excerpt from the webpage states:

> *"UE3 is Epic's next-generation technology, intended for games shipping on next-generation consoles and PCs."*

Features

- Skeletal animation system supporting up to 4 bone influences per vertex and very complex skeletons
- Full mesh and bone LOD support
- Animation is driven by an "AnimTree" - a tree of animation nodes including:
  - Blend controllers, performing an n-way blend between nested animationobjects
  - Data-driven controllers, encapsulating motion capture or hand animation data
  - Physics controllers, tying into the rigid body dynamics engine for ragdoll player and NPC[10] animation and physical response to impulses
  - Procedural skeletal controllers, for game features such as having an NPC's head and eyes track a player walking through the level
  - Inverse Kinematics solver for calculating limb pose based on a goal location (e.g. for foot placement)
- New node and controller types can be easily added for game specific control

[9] Similar to Function Controlled Effects described in this report.
[10] Non-Player Character

- Export tools for 3D Studio Max, Maya and XSI for bringing weighted meshes, skeletons, and animation sequences into the engine

Licence fee: $750,000
`http://www.unrealtechnology.com`

3.3 Conclusions

After thoroughly examining the above systems it was concluded that most systems come integrated in a game engine. While this might not be a problem it is not always desirable, for instance when a graphic system is already used, but lacks character support. The only existing stand-alone library was *Cal3d library*, but it was not among the high ranking systems. As one can expect the commercial engines are generally better than the open source systems.

New systems usually focus on skeleton animation in their design but does not bother much with outdated techniques such as mesh animation. It is becoming more important for a flexible library to include methods to extend and control skeletal animations, such as rigid bone attachment and inverse kinematics. This is the direction that Open Skeleton Library has taken. This, along with a comparision to the above systems, is presented in the next chapter.

4 Open Skeleton Library

This section starts with an introduction to the Open Skeleton Library, which also serves as a comparison to the systems presented in the previous section. It continues to explain the design approach and the essential elements of the library. After this new features are introduced and explained, finally the library structure is presented.

4.1 Introduction

The Open Skeleton Library is an excellent animation library that includes most of the features expected in a modern animation library and includes new improvements. It has a clear and modular design that facilitates solid structure to support future features. Since it is a stand-alone library, it is independent from any graphics platform, which makes it possible to include in any engine. Since it is not completed it still lacks many features.

The OSL can easily be compared to other systems even though they have had years of development time. The only other stand-alone library, Cal3d, hasbeen surpassed, much because it has not been properly renewed. Systems, like Nebula Device 2 or Unreal Engine 3, are more efficient since they use Vertex Shaders to skin their models, which OSL does not. Adding this feature would be the next logical step in the development of OSL. Compared to Unreal Engine 3, OSL still lacks integrated physics support, currently left to handled by the host system. Both systems feature a form of function controlled effects, which are very simular in purpose. OSL can also use these for additional purposes such as rigid bone attachments.

Features

- Skeleton animation controlled models
- Support for multiple formats easily extendable for new ones
- Weighted skinning with support for any number of weights
- Seemless animation blending
- Advanced function controlled effects for any part of the model
- Skeleton fitting makes it possible to use animation from other model formats
- Rigid bone attachment enables, for instance, a character to hold a weapon
- Stand-alone library makes it possible to include in any project

4.2 Design Approach

The Open Skeleton Library aims to surpass all existing character animation libraries in terms of useability and modularity. The focus is skeleton animation with features enabling extensions to the system.

The library design originated from the motion capture file format BVH[11], presented below. As this format lacks a mesh structure, this was also added to the base. This resulted in a library containing structures for

[11] BioVision Hierarchy file format, `http://www.biovision.com`

a *skeleton*, *keyframe animation*, *blender* and *mesh*. Support for new file formats are achieved by new parsers that are easily integrated to the library. This makes it relatively easy to include new formats. To facilitate the introduction of new features the library was made modular in its design and easy to manipulate. Extensions with popular features such as inverse kinematics or rigid bone attachment can be done through the use of *function controlled effects*.

The BVH format, that formed the basis for the library is a modelling format designed to store motion capture data. It has a hierarchial structure of joints and end effectors but lacks support for mesh data (The use of end effectors is described in Section 4.3.2). A typical excerpt from a BVH file is shown in Listing 1.

```
HIERARCHY
ROOT Hips
{
  OFFSET 0.0000 0.0000 0.0000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 3.7500 -0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
```

Listing 1: BVH sample

The other candidate to base the model structure on was H-Anim, an extensive file format that is intended as a standard for humanoid character models. It defines structures for joints, bones and end sites which correspond to end effector joints. Each object allows for advanced properties, such as orientation limits for joints and mass and other physical properties for bone segments. This format is primarily used for virtual worlds such as X3D[12]. While interesting, since it covers all parts needed for any animation purpose, it is too focused on humanoid models to be suitable for OSL.

4.3 OSL Basic Features

This section presents a walkthrough of the essential elements of the Open Skeleton Library. The features are similar to many animation libraries and for the basis of the system. New features are presented in section 4.4.

4.3.1 Model

The OSL model contains a skeleton structure and a mesh, which determines the pose and look of the character. It also has a number of animations and function based effects. They determine what actions the model is capable of performing. While it is not necessary for a model to contain all these things, all models must have a skeleton structure, since mesh animation is not supported. A quick overview of the models belongings are depicted in Figure 10. Furthermore, the model's update sequence is presented in Figure 11 which becomes more interesting as soon as other features of OSL are described.

[12]Extensible 3D, succesor to VRML (Virtual Reality Modelling Language)

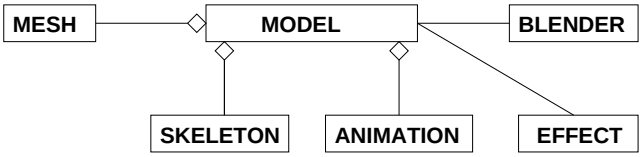


Figure 10: The Model structure

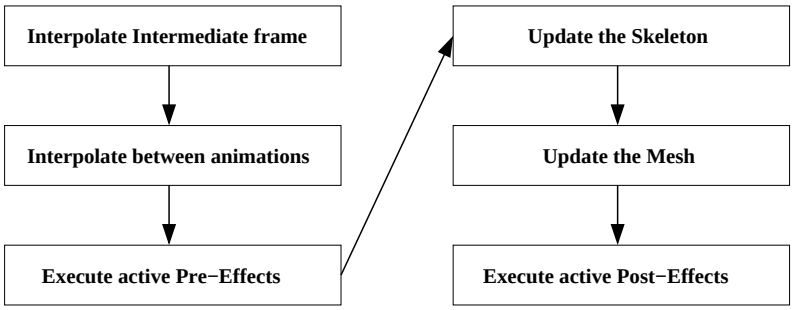


Figure 11: Sequence of the Models update

4.3.2 Skeleton Structure

The bone structure hierarchy consists of a number of joints, each connected to its parent by a length offset. This is the basic approach used in the BVH file structure and other modelling formats. Other ways of modelling the bones exist, for instance, by having both a structure for the bone and the joint. This would serve as a more realistic structure but with the cost of a higher complexity, but is generally not necessary. The joints have an offset to its parent and a rotation which the animation manipulates in order to set the skeleton in motions. The animation of the joints' offset can be used for muscle modellation and morphing effects.

Another feature inherited from the BVH format is the use of end effectors. An end effector is a rigid extension from the joint which can not rotate. This enables external tools to be attached to the body. The end effector keep its initial relative rotation to its parent. The root joint is the base of the skeleton structure and its offset indicate the model's translation. The structure of the bones are depicted in Figure 12.

4.3.3 Mesh

The mesh possess all basic properties, such as faces[13], vertex normals, texture coordinates and material properties, generally found in meshes and also supports weighted vertices. They are used to make the transitions between submeshes smoothly by connecting each vertex to several influencing bones as described in Section 2.4. The mesh currently supports any number of weights per vertex, but there is seldom need for more than four. The mesh is divided into submeshes, to facilitate the use of, for instance, different textures on a model.

[13] The faces define what vertices triplets that make up the triangles.

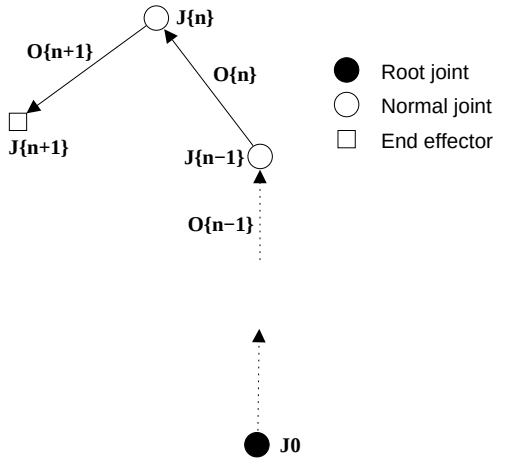


Figure 12: The skeleton structure

Procedual Mesh for BVH-skeleton

To show the easy extension possible for the library a simple procedual skinning is available for the BVH skeleton as there is no mesh defined in the BVH file. The mesh produced is a diamond shaped encapsulation of the bones with a girth proportional to the length of the bone. The result is a thicker and more lifelike figure than that of the earlier skeleton lines. This mesh, depicted in Figure 13, is similar to those used by the first skeleton-based computer games.

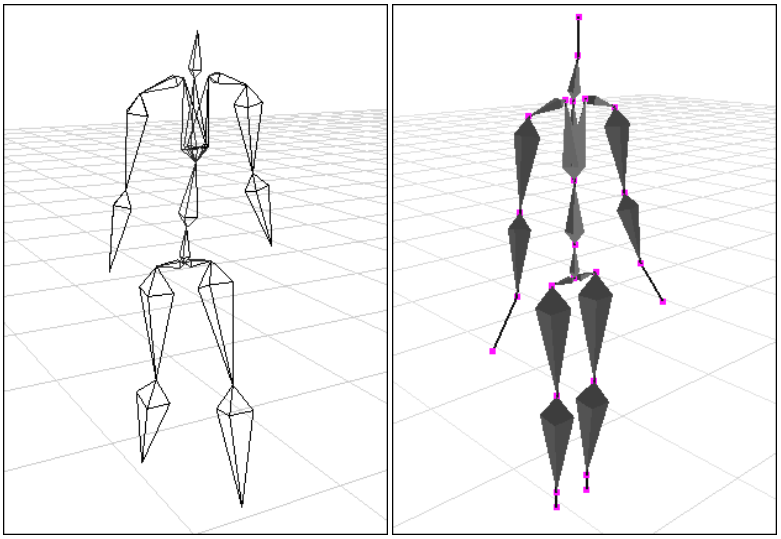


Figure 13: Procedual skinning of the bvh model

4.3.4 Animation

The animation system uses keyframe animation, described in Section 2.1, and an animation therefore consists of several keyframes each consisting of one rotation matrix and one offset vector for each bone in the skeleton. Each animation has a timer to keep track of the current keyframe position. The animation can either be looped or executed as a one-time-action. It is also possible to apply many animations to the same model at once, which is processed and blended by the Blender.

4.3.5 Blender

The OSL blender is responsible for two major steps in the animation process; interpolation between keyframes of one animation and incorporation of all active animations by interpolating these together to a single posture. To do this the Blender uses quaternions to do spherical interpolation, as described by [4], which are weighted by a value given by the user. Currently, the Blender will only blend whole models and not parts of them. This limits its useability in some situations. The problem with this method is described in Section 2.6.

4.3.6 OSL Parser: Model Importer

The OSL imports model formats through the use of different parsers. By simply writing a parser that extends the base parser, the library will gain a new format that it can use. This solution offers the possibility to keep the models in their original format and load them without any conversion back and fourth. Due to this, there is no need for OSL to house a native format.

4.4 New Features

This section presents the skeleton fitting scheme, as well as the function controlled effects. These are features new for this library and the basic theory for them are found in Section 2.9.

4.4.1 Skeleton Fitting

Skeleton fitting is performed by the `OSL_Parser` by taking two models, the target and source, and a joint map to determine what joints should be matched. This function will add a fitted animation to the target. Currently, a new map must be created for each new set of model formats that are to be used.

The implementation of this method works for some models, but strange artifacts are prominent in others. The reason for this is as yet undetermined, but adjustments can be made to compensate for this. Also, certain joints can be avoided when creating the map for the matching bones.

4.4.2 Function Controlled Effects

The Effects are implemented in OSL as a subclass of `OSL_Effect`, which forces it to implement the member function `execute(OSL_Model*)`. This is the function that will execute each update cycle of the model. It is tightly integrated with the model and has access to all its private members for total access. There are two sorts of Effects that can be added to the

model, Pre-Effects or Post-Effects. They are created in the same way but extra care must be taken for the Post-Effects since they will take place after the Skeleton and Mesh updates, (Figure 11), which means that the Model need to be correctly updated again.

The effects will typically execute every update cycle of the model, either prior to a update or as a Post-Effect. The use of post effects has the advantage in calculating the positions for multiple models only needing to do one update run through the entire model and adding the individual effects afterwards, skipping a lot of unnecessary recalculations. Of course it must be taken into consideration that changing a bone in a skeleton structure after its update means that a recalculation of its children must take place. The Pre-Effects are simply easier to create since no concern for specific update issues must be addressed.

In the case of Post-Effects, care must be taken to make sure that everything is updated correctly. For instance if the Effect influences a bone, it must make the effect of this alternation propagate to its children (if this is desired). An example appliction of this part of OSL is described in Section 5.2.

4.5 Further Improvements

Since the library is modular in its design and because the function controlled effects are an effective tool, it is easy to add new features to the library. A list of some possible improvements that can be made is found below:

- Inverse Kinematics
- Animation *bits* – blending control
- Level-of-detail
- Vertex shader driven skinning
- Integrated physics

5 Library Applications

5.1 Skeleton Fitting

The following example illustrates the use of skeleton fitted animations. The application shows the typical usage where a very good animation for a skeleton with few bones exists but the animation is to be used on an advanced model with a detailed skeleton with a mesh. In this demo, the MD5 model *fatty* from *Doom 3* is chosen as target model for animation. The source animation is taken from the the BVH model format, used for motion capture data. Target and source model in this scenario consists of 80 and 19 joints, respectively. The two models can be seen in Figure 14.

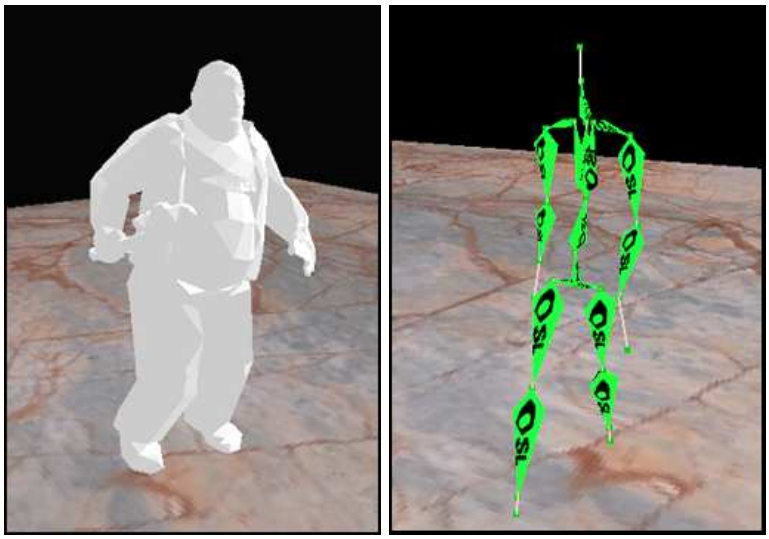


Figure 14: Target and source model

The process of transforming the animation from the source to the target starts with loading the different models with their corresponding models:

```
OSL_Parser_MD5 md5_p;
OSL_Parser_BVH bvh_p;

OSL_Model md5model = md5_p.parseModel("zfat/zfat.md5mesh");
OSL_Model bvhModel = bvh_p.parseModel("bvh/ballet.bvh");
```

To match the bones correctly, a map must is required. It contains an index at every position pointing to the matching joint index of the source model. If no matching joint is available, the default value will be -1.

```
std:vector<int> map;

for (int m=0; m<md5.getSkeleton()->getNbrOfBones(); m++) {
  std::string name =
    md5model.getSkeleton()->getBone(m)->getName();
  int bvhbone;

  if (name == "Body") bvhbone = BVH_HIPS;
  else if (name == "Lupleg") bvhbone = BVH_LEFTHIP;
  else if (name == "Rupleg") bvhbone = BVH_RIGHTHIP
  .
  .
  .
  else bvhbone = -1;
}

map.push_back(bvhbone);
```

This map is then used to apply the animation to the target model.

```
OSL_Parser::fitAnimation(md5model, bvhModel, 0, map);
```

This line of code adds the animation starting at index 0 from the `bvhModel` to `md5model`. The fitting of the bindpose is made by matching the target's bindpose to the source's bindpose which can be seen in Figure 15, from left to right, the default bindpose of target, the source's bindpose and finally the target's fitted bindpose. The fitting is now complete and the animation is loaded and usable by the `md5model`. A screenshot from *fatty* dancing ballet is shown in Figure 16.

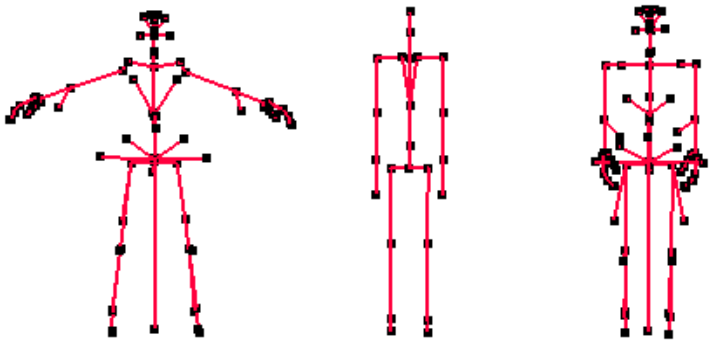


Figure 15: The steps of skeleton fitting

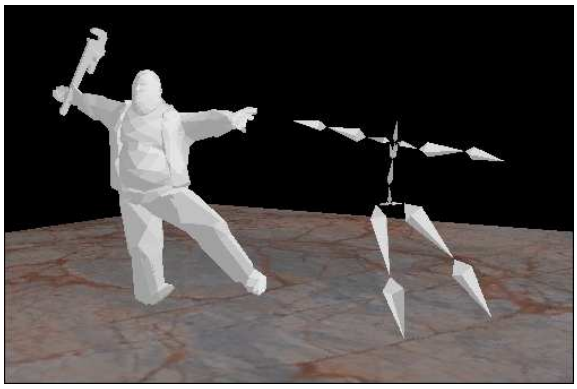


Figure 16: Fatty zombie dancing ballet

5.2 Randomized Crowd

The object of the *Randomized Crowd Demo* is to simulate individual characteristics for every person in a large crowd. It can be achieved by slightly rotating a person's bones in different ways by the use of the OSL:s Effect module. First a proper Effect, *randomised* in some way, is created, which is added to a model to make it appear slightly different. This is done for every person in the crowd.

A simple effect which can set the rotation of individual joints is implemented as:

```
OSL_Effect_RotJoint::OSL_Effect_RotJoint(int jIndex,
  const OSL_Vector3& axis, float angle)
  {
    jointIndex = jIndex;
    OSL_Quaternion q;
    q.set_rotate_axis_angle(axis, angle);
    rotation.set(OSL_Matrix3(q));
  }

  void OSL_Effect_RotJoint::execute(OSL_Model* model) {
    vector<OSL_Matrix3*>* store = &(model->matrixStore[0]);
    (*store)[jointIndex]->set(rotation * *(*store)[jointIndex]);
  }
```

Listing 2: 'effect_rotjoint.cpp'

For total control when writing Effects, they are allowed access to the private members of a model. Here it uses the models `matrixStore` which is the structure all joints rotations are stored before sending them to the update function. In OSL, an Effect can either be added as a Pre-Effect, influencing the joint just before being sent to update the skeleton, or as an Post-Effect influencing the joints after the update. The easiest (but not the most efficient, see Section 2.9.2) is to use the Pre-Effects. This is done by assigning an Effect to the model like:

```
  OSL_Effect_Bone fx(9, OSL_Vector3(1,0,0), PI/6.0);
  model.addPreEffect(fx);
```

5.3 Rigid Bone Attachment

To illustrate how effective the function controlled effects are, they have been used to implement rigid bone attachments for OSL. The attachment is implemented through a Post-Effect, taking an `OSL_Model` as the attachment, and connecting it to a designated bone. It is then added to a target model where it's `execute` function replaces the attached model's root offset and rotation found at the target's designated bone. Since it is a Post-Effect it will execute after the target model has performed its updates, thus, putting it in the right place. The effect used can be seen in Listing 3.

```
void OSL_Effect_Attachment::execute(OSL_Model* model) {

    int par = model->skeleton->getJoint(jointIndex)->getParent();
    *(attachedModel.matrixStore[0][0]) = rotation *
      model->skeleton->getJoint(jointIndex)->getGlobalRotation();
    *(attachedModel.vectorStore[0][0]) =
      model->skeleton->getJoint(par)->getRootOffset();

    attachedModel.skeleton->update(attachedModel.matrixStore[0],
      attachedModel.vectorStore[0]);
    attachedModel.updateMesh();
}
```

Listing 3: 'effect_attachment.cpp'

The *trick* is, as in the randomized crowd application, to communicate with the model update through the use of its matrix and vector stores. The result using a BVH model with a MD5 chainsaw model attached can be seen in Figure 17.

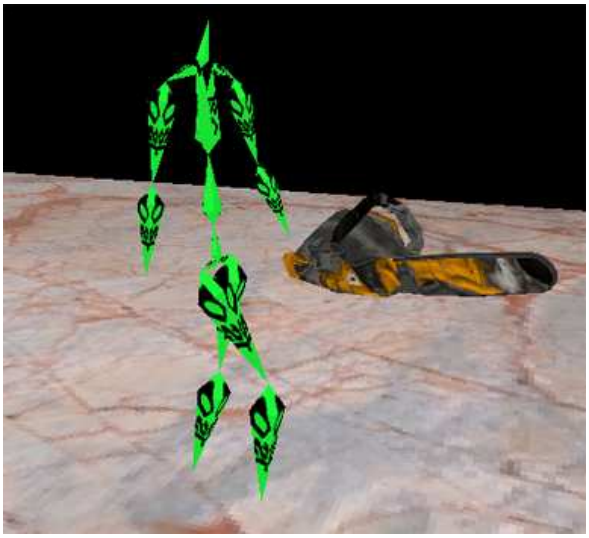


Figure 17: BVH model carrying a MD5 model chainsaw

.

6 Conclusions and Results

The goal was to achieve a stand-alone animation library that is more complete than any other existing system. In many ways this was achieved. The Open Skeleton Library has a sound design capable of including many features. All implemented animation techniques in the library are smoothly integrated with each other and can be used in a smooth way. All basic character animation methods, such as level of detail and inverse kinematics, are not implemented. OSL provides a structure which makes it possible to add these features with little trouble. The performance of the library can be improved by implementing vertex shader driven skinning.

The new features, skeleton fitting and function controlled effects, were succesfully implemented. Through the Effects the same model can be used repetedly with individual characteristics, preventing the appearance of the cloning effect often seen in games. The Effects can also be used to easily add new arbitrary methods of control. The skeleton fitting is an excellent tool for applying animations from one model to another. This is especially useful when motion capture data, which is obviously done for a human skeleton, should be applied to a non-human model.

Although not complete the Open Skeleton library has the potential to be one of the leading open source character animation libraries. A clear design and extensible structure makes it ideal as a modular extension in, for instance, a game engine.

7 References

References

[1] Watt, Alan and Policarpo Fabio,
3D Games - Animation and Advanced Real-time Rendering Vol. 2, 2003
Pearson Educational Lmt, pages 358-362.

[2] Sébastien Dominé
Mesh Skinning
*n*Vidia 2005

[3] Pablo de Heras Ciechomski
Parametric Dynamics - A method for addition of dynamic motion to computer animated human characters
Lund University of Technology 2001

[4] Paull, David B.,
Programming Dynamic Character Animation, 2002
Charles River Media, inc, pages 164-165.

[5] BioVision Incorporated
`http://www.biovision.com`

[6] hanim Humanoid Animation Workgroup
`http://www.h-anim.org/`

Game engines and animation libraries

Cal3d - `http://cal3d.sourceforge.net`
Nebula Device 2 - `http://www.nebuladevice.org`
Irrlich Engine- `http://irrlicht.sourceforge.net/`
Cipher Game Engine - `http://www.cipherengine.com/`
Torque Game Engine - `http://www.garagegames.com/`
Truevision 3d - `http://www.truevision3d.com`
Unreal Engine 3 - `http://www.unrealtechnology.com/`