

# **Skinned Animation**

*Author: Max Wagner, mwagner@digipen.edu*

## Pre-Reqs

C++, Microsoft Visual Studio IDE, general computer graphics knowledge and 3D math.

## Overview

Animation is a rather large topic, and as such it would probably require a large tome to cover it all (if that were even possible). Currently, I'm not looking to write a large tome, so let's narrow the topic down a bit. Here's what I will be discussing, and what I won't be discussing.

What I will discuss:

- Some theory behind articulated skeleton hierarchies (bones).
- Rotations and quaternions.
- Minimal data that should be contained in a model file format (custom or not).
- Basic data structures required to define a custom animation framework.
- Implementation of skinning in assembly DirectX Vertex Shaders, version 1.1.

What I won't discuss:

- .X file format, or ID3DXMesh. While my knowledge of the DirectX Utility animation framework is mostly cursory (looking through the documentation and samples, but never actually implementing it myself), I have to say that I tend toward home-grown solutions. This is for educational purposes, as well as practical.

## The Hierarchical Model

### A First Attempt at an Algorithm

Rendering a standard model is very straightforward. We set the vertices, the world transform, and then render the triangles. The vertices that we specify are in object space, and somewhere in the hardware, these vertices are getting transformed by the world transformation matrix that was set prior to the draw call. Lighting will usually be applied in the world space, and then vertex data will be output to the clipper, undergoing a final transformation to clip space. What's important to note is that *all* the vertices are undergoing the identical transformation.

This same process cannot handle an animated mesh. One possible attempt to solve the problem of the animated mesh would be to actually use multiple meshes, one for each frame. Then, you would calculate which frame of the animation to use, set those vertices (still in object space), and then continue as above. The problem with this approach, of course, is that it requires vast amounts of memory. Imagine one model using just 500 triangles. If one animation loop lasted 3 seconds, and the resolution of the frame

sampling were in half-seconds, then you would have to store 6 times the vertex count. This quickly becomes unacceptable.

## Skinning

While there is no one way to handle an animated mesh, one very common technique that is gaining even more popularity due to its easy translation to hardware-based implementation is that of vertex skinning, alternatively known as using a matrix-palette or indexed vertex blending.

This technique requires only one set of vertices defining the model. In addition, each vertex contains two, three, or sometimes more *indices*. These indices index into a list of *bones* (the *skeleton*) which contain transformation data at different *key frames*. In addition to the indices, each vertex also contains as many *weights*, defining the percentage amount of the vertex's dependence on the corresponding indexed bone transformation.

More formally, given:

- $N$  bones affecting a given vertex  $\mathbf{v}$ ;
- the vertex's position  $\mathbf{p}$ ;
- the vertex's bone index array  $\mathbf{b}$  and weight array  $\mathbf{w}$  where the  $i^{\text{th}}$  weight corresponds to the  $i^{\text{th}}$  bone index;
- the model's skeleton  $\mathbf{S}$ , which can be indexed to provide a reference to a bone;
- the bone is a function of time  $\mathbf{t}$ , outputting a 4x4 transformation matrix;

a vertex is transformed (animated) using the following equation:

$$v_{\text{animated}} = \sum_{i=0}^{N-1} w[i] * S[b[i]](t) * p$$

This formal definition of the animated transformation of a vertex reveals specifically the data and logic required to perform vertex skinning.

## Key Frames

Above I described the bone as a function of time. The function, however, is not a regular old analytic function, but rather a function that is handled by interpolation of key frames, where a given time  $\mathbf{t}$  falls into a given *interval*, allowing for interpolation between the two key frames which book-end that interval. We will discuss exactly how this interpolation occurs later when we get into quaternions, but for now you can think of it as some sort of linear interpolation that occurs between the transformations stored at key frame  $\mathbf{i}$  and  $\mathbf{i+1}$ .

## The Bones

Let's examine the skeleton a little more closely. Consider a simple animation sequence representing the bending of a human arm. Think of the (simplified) skeleton of the arm

as containing an upper arm, forearm, hand, and fingers. A vertex located near the elbow (between the upper arm and forearm) might rely equally on the upper arm bone and the forearm bone. That vertex would contain indices which correspond to these bones, and according weights (probably 0.5 and 0.5).

Above, in the formal definition of an animated vertex, we stated that the bone is a function of time, outputting a 4x4 transformation matrix. Just what is the transformation that it outputs?

Consider our hypothetical arm animation as an arm doing a bicep flex: the arm is initially outstretched; the upper arm does not move, while the forearm rotates upward until slightly past 90 degrees with the upper arm; the hand meanwhile clenches into a fist. The transformation required to describe the upper arm is clearly just the identity matrix. But what about the forearm, the hand, and the fingers? In what space should we attempt to describe their motion?

### Local Space

Consider that each bone in the skeleton (aside from the root) contains one (and only one) parent bone. The motion of the forearm is wholly dependent on the upper arm, and, though in our example we've only been discussing the arm, the upper arm clearly relies on its parent, the clavicle (or perhaps the neck).

An individual bone's motion, while dependent on the parent, can be visualized and interpreted separately. For example, I can twist and rotate my foot at the ankle. I can also do this while lifting my leg, but the overall motion (from the point of view of my foot/ankle) remains the same. Therefore, it is useful to describe this motion in a manner which is independent of the parent. Later on, we will see that this local space is not only useful and intuitive, but "correct" for the purposes of interpolation; describing the bone transformations in other spaces (such as model space) yield incorrect interpolation.

A bone's transformation into model space is thus defined recursively as:

- If the bone has a parent:
  - The bone's local transformation  $\mathbf{L}$  transformed by the parent's transformation  $\mathbf{P}$ , i.e.  $\mathbf{P} * \mathbf{L}$ .
- Otherwise (if the bone has no parent):
  - The bone's local transformation  $\mathbf{L}$ .

Given this definition, we can say that to get to the bone's local space, we multiply on the left by the inverse of the parent's transformation (if there is a parent).

For various reasons, we will limit the allowable transformations on a bone to be translation and rotation (specifically, we will not allow scaling). Let's break up the bone transformations into these two components and analyze each separately.

### Bone Local Rotation

The notion of the bone's local rotation is really quite simple, now that we have the definition of a bone's transformation to model space as given above. Quite simply, if we imagined that we were transforming the points of the bone, we would apply the bone's local rotation, followed by the parent's full (to-model-space) rotation.

### **Bone Local Translation**

First, let's adopt the convention that an untransformed bone lies along the y-axis of its local coordinate frame. Then after rotation, and assuming the bone is rigidly connected to the tip of its parent bone, the bone is then translated along its y-axis by the length of its parent bone. Sometimes, it is useful to store bone offsets, in the case that we don't want a given bone to be directly connected to its parent's tip. This displacement is likewise specified in the bone's local space.

Thus, the bone's local 4x4 transformation  $\mathbf{L}$  can be fully described by its columns as

$$\mathbf{L} = [ \mathbf{R}_0 \mid \mathbf{R}_1 \mid \mathbf{R}_2 \mid \mathbf{T} ]$$

where  $\mathbf{R}_i$  is the  $i^{\text{th}}$  basis vector of the bone's local rotation, and where  $\mathbf{T}$  is the vector offset from the parent bone along the y-axis, combined with any displacement.

[DRAFT – UNDER CONSTRUCTION]