Deformable Skinning on Bones

Bent Dalgaard Larsen Kim Steen Petersen
Bjarke Jakobsen

17th December 2001

Abstract

Applying skin to a model is a relatively simple task to implement. Nonetheless it seems that no good resource exists that describes both the concepts and math necessary to understand and implement skinning. The intention of this article is an attempt to give a thoroughly description of the theoretical and mathematical background of skinning.

1 Introduction

A real-time character created using a hierarchy of bones can have geometry attached in several ways. An easy method is to attach cylinders or other primitives to each of the bones, but this is a very rough and not visually very pleasing solution. A more sophisticated method is to use a polygon mesh for skinning. But as the bones a rotated and translated the skin need to be deformed. The first solution would be to make a pointer from each vertex to a bone a let it be fixed relative to this bone. This solution works but problems arise near joints. A better solution is to make each of the vertices be affect by several bones by a certain percentage. The rest of this article will describe this particular solution and the math covering this method.

2 Theory

Initially we have the position of the vertex $v^{(0)}$, the position and rotation of the i'th bone $b_i^{(0)}$ and $R_i^{(0)}$. Jointly $b_i^{(0)}$ and $R_i^{(0)}$ could also be written as a

name	description
b_i	current position of bone i
R_i	current rotation of bone i
T_i	current transformation of bone i as matrix
n'_i	offset normal for v in bone i coordinate system
v'_i	offset vector for v in bone i coordinate system
v	current vertex position
$v^{(0)}$	initial vertex position
$b_i^{(0)}$	initial position of bone i
$R_i^{(0)}$	initial rotation of bone i
$T_i^{(0)}$	initial transformation of bone i as matrix

Table 1: Description of variables

4×4 matrix, but for the purpose of simplicity and since $R_i^{(0)}$ could be either a quaternion or a 3×3 matrix we will write them separately. We also know the weight w_i with which the bone affects the vertex v. We denote the total number of bones n. It is noted that the weights of all bones affecting a vertices is summed to 1 i.e. $\sum_i^n w_i = 1$. Our purpose is now to find v when the bone position b_i and rotation R_i changes for each bone. For that purpose we use an intermediate vector v'_i that we calculate initially. v'_i is the position of v in bone i coordinate system. Disregarding the weights, the position of v in any bone coordinate system will always be fixed. v'_i can now be found in the following way.

$$v^{(0)} = b_i^{(0)} + R_i^{(0)} v'_i \Rightarrow v'_i = {R_i^{(0)}}^{-1}(v^{(0)} - b_i^{(0)}) \tag{1}$$

Now it is possible to express v as a function of the current bone position b_i and rotation R_i and the calculated offset value v'_i.

$$v = b_i + R_i v'_i \tag{2}$$

But since v is affected by n bones with weights w_i the position is calculated as

$$v = \sum_i^n w_i(b_i + R_i v'_i) \tag{3}$$

In some situations it is desirable to express this as a function of $v^{(0)}$ and a matrix multiplication. We will now express the position and rotation as a 4×4 matrix $T_i^{(0)}$. Rewriting (1) we get

$$v^{(0)} = T_i^{(0)} v_i' \Rightarrow v_i' = T_i^{(0)^{-1}} v^{(0)}$$

Rewriting (3) we get

$$v = \sum_i^n w_i T_i v_i' = \sum_i^n w_i T_i T_i^{(0)^{-1}} v^{(0)} = \sum_i^n w_i M_i v^{(0)} \quad (4)$$

where

$$M_i = T_i T_i^{(0)^{-1}} \quad (5)$$

Often it is necessary to calculate the normal in order to get correct lighting. Using (1) and (3) the normal will be calculated by just removing the translation part b_i yielding

$$n^{(0)} = R_i^i n_i' \Rightarrow n_i' = R_i^{(0)^{-1}} n^{(0)} \quad (6)$$

and

$$n = \sum_i^n w_i R_i n_i' \quad (7)$$

Again rewriting this as a function of the initial normal and a matrix multiplication yields

$$n = \sum_i^n w_i R_i R_i^{(0)^{-1}} n^{(0)} \quad (8)$$

The rotation $R_i R_i^{(0)^{-1}}$ can also be expressed using the inverse transpose of the same transformation matrix as used in (5). A thorough explanation of this relation can be found in [2]. Rewriting (8) yields

$$n = \sum_i^n w_i M_i^{-1^T} n^{(0)} \quad (9)$$

3 Implementation issues

It is well known that a rotation can be described both as a quaternion and using a rotation matrix. Quaternions have several advantages compared to rotation matrices [3]. When implementing skinning in software it will therefore be advantageous to implement calculation of vertices and normals using equation (3) and (7) and using quaternions.

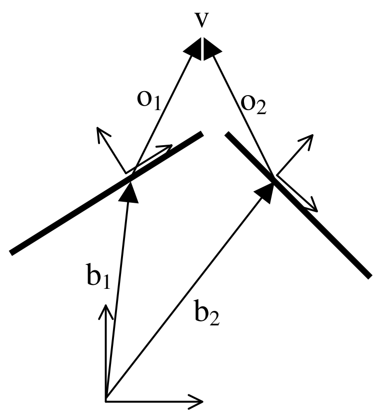

Figure 1: Illustration of vectors

As the model that need to be skinned grow in size the amount of math operations necessary grow rapidly. All calculations need to be calculated each frame and usually it will not be possible to use any frame-to-frame coherence. Furthermore the calculations are trivial and it is therefore evident that these calculations are suited for implementation in hardware. Modern low-end graphics card supporting Vertex Programs[1] are capable of calculating all matrix transformations directly in the graphics processor using a simple but limited assembly language. Graphics processors only support matrix transformations and it is thus necessary to use equation (4) and (9) when implementing skinning in hardware. Furthermore equation (9) is advantageous since it uses the same matrix as (4) because the graphics cards have limited space for storing matrices [5].

4 References

[1] Lander, Jeff : *"Skin Them Bones"*, GameDevelopers Magazine, pp. 11–16, 1998
http://www.darwin3d.com/gamedev/articles/col0598.pdf

[2] Turkowski, Ken : *"Properties of Surface-Normal Transformations"*, in Andrew Glassner (editor), Graphics Gems I, Academic Press, Inc., pp. 539–547, 1990.
http://www.worldserver.com/turk/computergraphics/NormalTransformations.pdf

[1] Vertex Programs are sometimes refered to as Vertex Shaders.

[3] Möller, Thomas and Haines, Eric : *"Real-Time Rendering"*, pp. 23–52, 1999

[4] Eberly, David H. : *"3D Game Engine Design"*, pp. 356–358, 2000

[5] Dominé, Sébastien : *"Mesh Skinning"*, 2001, http://www.nvidia.com