Character Animation with Triangle Meshes and Motion Capture Data

Robert Rose

May 3, 2004

Abstract

This brief article describes an approach to rendering articulated characters using triangle meshes and motion capture data. It is assumed the reader is already familiar with animating bone structures, such as Biovision BVH files.

1 Data Format

1.1 Character Model Data

The character model that we are interested in rendering is given as a bone structure and a triangle mesh. A bone structure is a hierarchy of joints. Each joint is specified relative to it's parent by an offset and a rotation. A triangle mesh is a large collection of triangles that when drawn together "mesh" together to form surfaces. Triangle meshes are typically described as an array of vertices and an array of 3-tuples that are indices into the vertex array.

To animate the character, we want to tie the triangle mesh to the bone structure so that as the joints move, the triangle mesh moves with it. In order to do this we associate each vertex in the triangle mesh with a joint in the bone structure. As the joint moves, we move all of the joint's vertices with it.

It is important to note that vertices–not triangles–are associated with joints. If triangles were associated with joints instead of bones then each joint would need its own triangle mesh to go with it, and obvious visual artifacts such as tearing can occur as the joints rotate (such as when an elbow bends 90 degrees). Because vertices are associated with joints, the triangles will squash and stretch to prevent tearing. This approach can

cause noticeable artifacts also (such as when an elbow *twists* 180 degrees), but typically only when a joint moves in an unrealistic manner.

1.2 Motion Data

Motion data is given as a series of joint rotations, plus a translation for the root joint.

2 Animation

2.1 Preparation

Typically the triangle mesh for the character model will be given in world coordinates, however each joint in the bone structure of the character is given in coordinates relative to it's parent. Before we begin animating, we need to transform each vertex in the character triangle mesh into the coordinate system of the joint it belongs too. This can be found by multiplying the *inverse* of the joint's transformation matrix by the vertex in world coordinates.

Recall that when animating a bone structure we begin at the root node, applying the joint's translation and rotation to it. We then walk down the hierarchy, applying each new joint's translation and rotation, and pushing and popping the current transformation matrix in case a joint has multiple children associated with it. To find each vertices' position in the coordinate system of it's joint, we go through this recursive process when we load the model. At each joint, we will multiply all of the joint's vertices' by the inverse of it's transformation matrix.

2.2 Render

On each time step in the motion data, we need to find the new values for the character model triangle mesh in world coordinates. We do this by recursing through the bone hierarchy and multiplying all joint vertices' (in "joint coordinates", as determined from the preparation step above) by their joint's transformation matrix.

Once all vertices have been transformed back to world coordinates we step through the triangle list and draw each triangle.

2.3 Lighting

Although our triangle mesh data may have contained surface normals for each triangle, they are useless to us once we begin animating. On each time

step in the motion data, surface normals will need to be re-calculated. This can be accomplished by creating two vectors for each triangle that represent the sides of the triangle, taking their cross product, and then normalizing the result.

3 Conclusion

Animating a triangle mesh using a bone structure is an expensive process. If a certain degree of visual artifacts are acceptable, then bone / triangle meshes should be avoided–characters should be modeled using solid object techniques.

After having gone through this exercise if became very clear to me why many video games still use low poly-count characters but high poly-count terrains and objects. I'm curious if vertex shaders could be abused to handle character animation, offloading the expensive matrix multiplications necessary to transform a character model from joint coordinates to world coordinates to the graphics hardware.