CS 560 - Animation Project 1

Egemen Koku

How to Run and Interact With The Project

To run the project, simply run “AnimationAssignment1.exe” in the same directory as the solution. I takes a few seconds to show anything after launching (I don’t have parallel threads so it acts like a loading time while I’m loading the model).
W,A,S,D controls the camera direction, Q and E raises and decreases camera respectively.
Holding Left Shift makes the camera go faster.
Pressing R button makes the camera snap back to its original position.
To rotate the camera, hold down Right Mouse Button and move the mouse.

Model Information

I’m using FBX models as an example model. The example model I’m using is Tad, which has a fairly complex structure.

FBX Data Structure

Fbx has nodes called FbxNode and every node holds the Transformation information (both global and local) inside. The scene created in FBX is basically made of a tree of nodes. I created a structure like that in my assignment.

Internal Data Structure

Each control node in the skeleton is described with a class called SkeletonNode in my code base. In Model Manager, I go through each FBX node in the scene and create it’s equivalent SkeletonNode. Each SkeletonNode contains their own children so they are held as a tree of nodes. This way, I just need to keep the root SkeletonNode in the Graphics Manager.

During render call, Graphics Manager calls Render function of the root node and starting from the root node, each node calculates their own Transformation data, sends it to the shader, calls draw of the mesh and then calls draw function for each of their children in a depth-first manner. In the end, every node draws a cube in their joint position and a line between themselves and their parents.

Animation Data

When I'm creating my SkeletonNode tree in Model Manager, I recursively call InsertNode for all the nodes starting from the root node. Each SkeletonNode holds their own animation data in a map.

For each node, I loop through the frames of the animation (we can get total frame from FbxTakeInfo class). FbxNode has a function called EvaluateLocalTransform that returns the local transform value of the node. I get the transform matrix of each node using EvaluateLocalTransform and get translate and rotate information from it (scale is 1 for all nodes).

From the rotate value, I create a Quaternion and then using that Quaternion with the translate value I get from the matrix, I create a VQS structure. I have a map in each SkeletonNode that holds <KeyFrame, VQS> pairs in it so I store the VQS with the frame counter I'm on at that point. I do this for all the frames in the animation so after the creation of the tree, I have the VQS of each node stored in a map.

Animating the Model

In the draw call, before drawing the entire tree recursively, I calculate which key frame I'm on at that point and the interpolation delta value based on how much time I spent in that key frame. With the draw call, I send these current frame and interpolation value throughout the all the nodes. In each node, it does a lookup from the map and gets the VQS related to the current frame and uses interpolation using the interpolation value to find the actual transform point in the curve.

The root node contains its world transformation value and all the children have the local transformation value. To calculate where every node will be in the world space, I use the interpolated VQS value and concatenate it with node's parent VQS to transform it to the world space. Since I go through all the nodes in a depth first manner, every node finds its place in terms of world coordinates.

After the concatenation, resulting VQS's translate value contains the final position of the node so I just call glm::translate using that vector to get matrix equivalent of the world space and send that data to the shader to draw the joint with respect to its parent's position.

Interpolation

Since I'm using VQS, I need to interpolate for the translate vector and rotation quaternion (since all the scale data is 1.0, I'm just using that value for the scale). As for the interpolation value, I'm using the value I calculate at the graphics manager

For translate vector, I'm using Linear Interpolation because it is basically a continuous line.

```
(1.0f - delta) * vqs1.translate + delta * vqs2.translate;
```

For Quaternions, I'm using a combination of Slerp and Lerp. I'm using Lerp if the angle is too small and Slerp for the rest of it. For the interpolation, I'm actually using an algorithm I find from Will Perone's website. But basically, it's something like this:

```
if (dot < 0.95f)
{
    float angle = acosf(dot);
    return (q1*sinf(angle*(1 - t)) + q3*sinf(angle*t)) / sinf(angle);
}
else // if the angle is small, use linear interpolation
    return Lerp(q1, q3, t);
```

After those two interpolations, I just create the middle VQS with those interpolated translate and rotation values.