Design Document - Assignment 4

Naveen Ali
UT EID: nna668

Tarun Gunampalli
UT EID: tg25522

March 2022

1 Bone Picking

In Gui.ts, we can create a function that will convert the mouse position to a ray in world coordinates. We can do this by taking the mouse coordinates and converting them to NDC. We can take the x coordinate and convert it to NDC by multiplying it by 2 and dividing by (width - 1). We can take the y coordinate and convert it to NDC by doing 1 - (2 * y) and dividing that by the viewPortHeight. We can then create a vector with these NDCs and then multiply by the inverse of the projection and the inverse of the view matrix. We now have a direction for the ray in world space. The origin of the ray will be the eye position. We can call this method in the dragStart method to obtain the mouseRay. We can then use this mouseRay to find an intersected bone. We can create a method called findBone. This will go through each mesh in the scene and it will go through each bone in the mesh. We can then call another function to check if the ray and the bone intersect. We will want to keep the earliest intersected bone so we will also need to check at what t did the ray intersect the current bone. When checking for if a mouse ray intersects with a bone. We first need to obtain the rotation matrix that aligns the bone with the y axis, so that we can get the bone and the ray into the xz plane. We can then create a circle intersect function that checks if the ray intersects the cylinder. In order to do a lot of this, we added an intersectedBone instance variable to the GUI class that allows us to store all the information about the bone that we're hovering over. For visualizing a highlighted bone, we can create a cylinder class. The cylinder class will store the coordinates of our vertices and set the indices array. We can then create 2 new shaders, a vertex and fragment shader for the cylinder. The vertex shader will set the gl_Position by multiplying the vertex position by a scaling matrix, a rotation matrix, and a translation matrix, and then the view matrix and the projection matrix. We can then create a renderpass in app.ts. We then create a initCylinder function to setup the renderpass. We add the vertex positions as the attribute for the cylinder. We can then add the projection matrix, view matrix, scaling matrix, rotation matrix, and translation matrix to the renderpass. We can create a method in gui.ts, that will calculate the scaling, rotation, and translation matrices.

2 Updating the Skeleton

In the drag function, if the left button is pressed, and an intersected bone does exist, we can obtain the rotation axis by taking the normalized cross product of the camera's forward vector and the negative mouse direction. We can then obtain a rotation quaternion by using the fromAxisAngle function and passing in the rotation axis we calculated. We can then create a new recursive function to rotate a bone and propagate that rotation to all children bones by passing the function the quaternion we just obtained. For rolling, in the function onKeyDown in Gui.ts, if the left or right key is pressed, we can calculate the tangent line to the bone and create a quaternion using the the tangent line and then use our recursive bone rotation function.

3 Linear Blend Skinning

All of the things we need is already in the shader so we need to alter sceneVStext. We sum the weight of the bone * vertex position. We then apply the bone's translation and rotation matrices and then we multiply by the view matrix and the projection matrix.

4 Texture Mapping

For texture mapping we plan to use similar code to how we did cube mapping in assignment 1. In RenderPass, we will pass the texture map into the shader and then use the colors from the texture map in combination with lighting calculations to determine the color of the object.