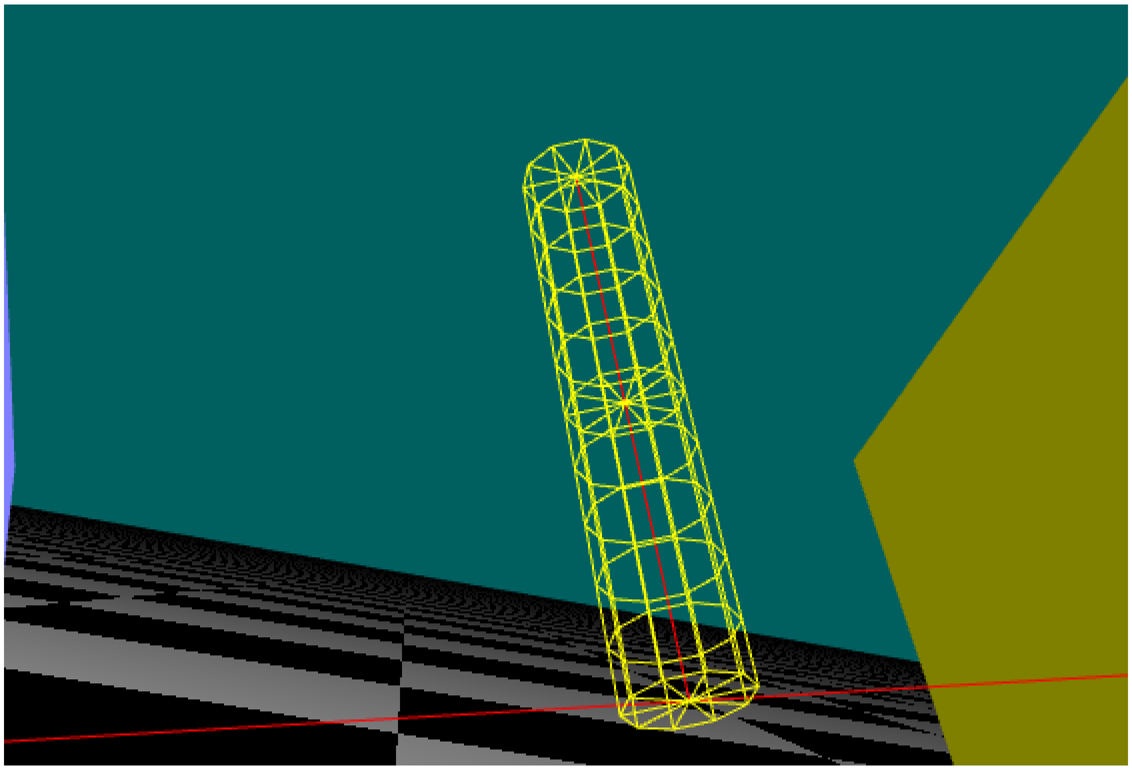
Bone Picking

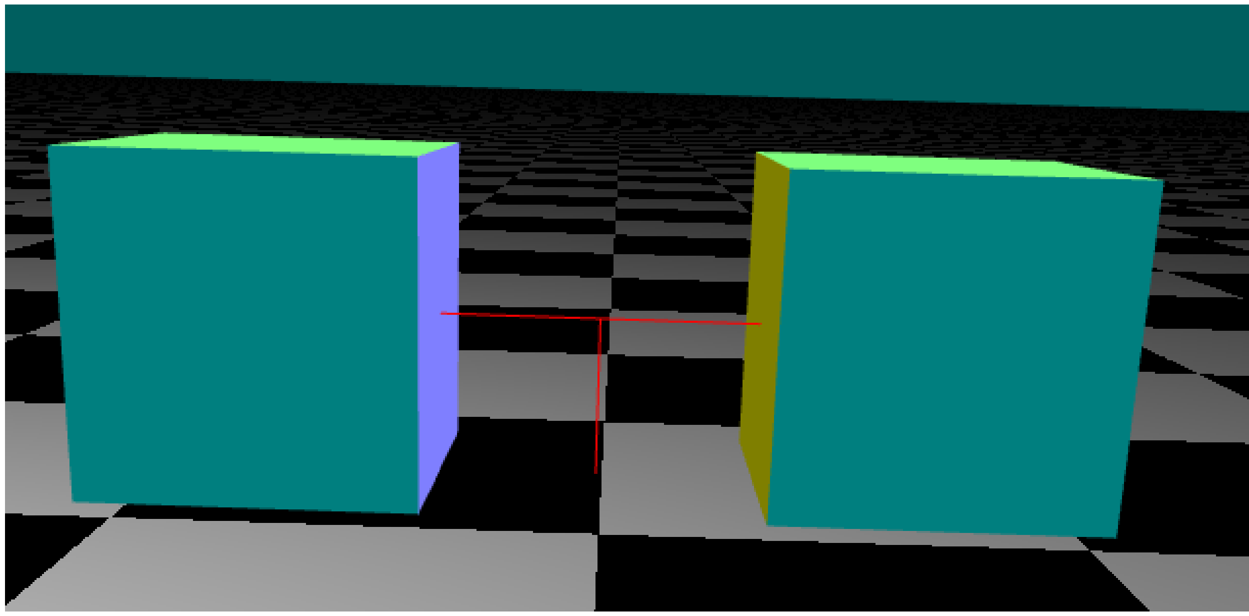
- Getting p and v
    - I converted the mouse cursor movement to world coordinates by taking the mouse offset x and mouse offset y, converting them to ndcs, then multiplying the inverse view matrix and inverse projMatrix by the ndcs. I inversed the w component and returned. This gives the mousePosWorldCoords
    - I get a ray relative to the camera, v, by taking mousePosWorldCoords' difference with p, the camera's position.
- Finding the closest bone intersection
    - Now that I have the ray, v, and its origin, p, of the mouse cursor, I loop through all bones and find the closest cylinder around the bone that the ray intersects.
        - I check if the ray intersects a cylinder around a bone by getting the cylinderStartPos (bones position), cylinderEndPosition (bones endpoint), the radius of the cylinder (currently defined as .1), p (camera position), and v (where the mouse is pointed to in world coordinates relative to the camera position).
        - I solve the solutions of a parametric equation representing the cylinder, which is defined here: https://iquilezles.org/articles/intersectors/, and if it has a solution, then it intersects.
    - The intersection with the lowest t value (distance from camera origin to cylinder) is kept track of and will be chosen for highlighting.
- Highlight the closest intersected bone
    - If no intersection was found, then highlights are removed. Else, I highlight the bone that was intersected.
    - Highlights are done by surrounding the bone with a gridded cylinder.
        - I construct an NxN gridded unit cylinder (N is hardcoded to be 10) of length 1.
        - I add caps on the ends and middle for visual appeal
        - I calculate its scale matrix using the length of the bone, translation matrix using the position of the bone, and rotation matrix using the bone's tangent, normal, and binormal.
        - In a custom shader, I transform the position of each vertex via transMat * rotMat * scaleMat * vertPosition, which scales and orients the unit cylinder around the target bone.
- Side Note
    - I have some custom logic I use for whether or not a bone highlight stays for a better user experience. The bone is only deselected if the user clicks off of it or hovers over a different bone and isn't dragging.
- Results

- ○ 

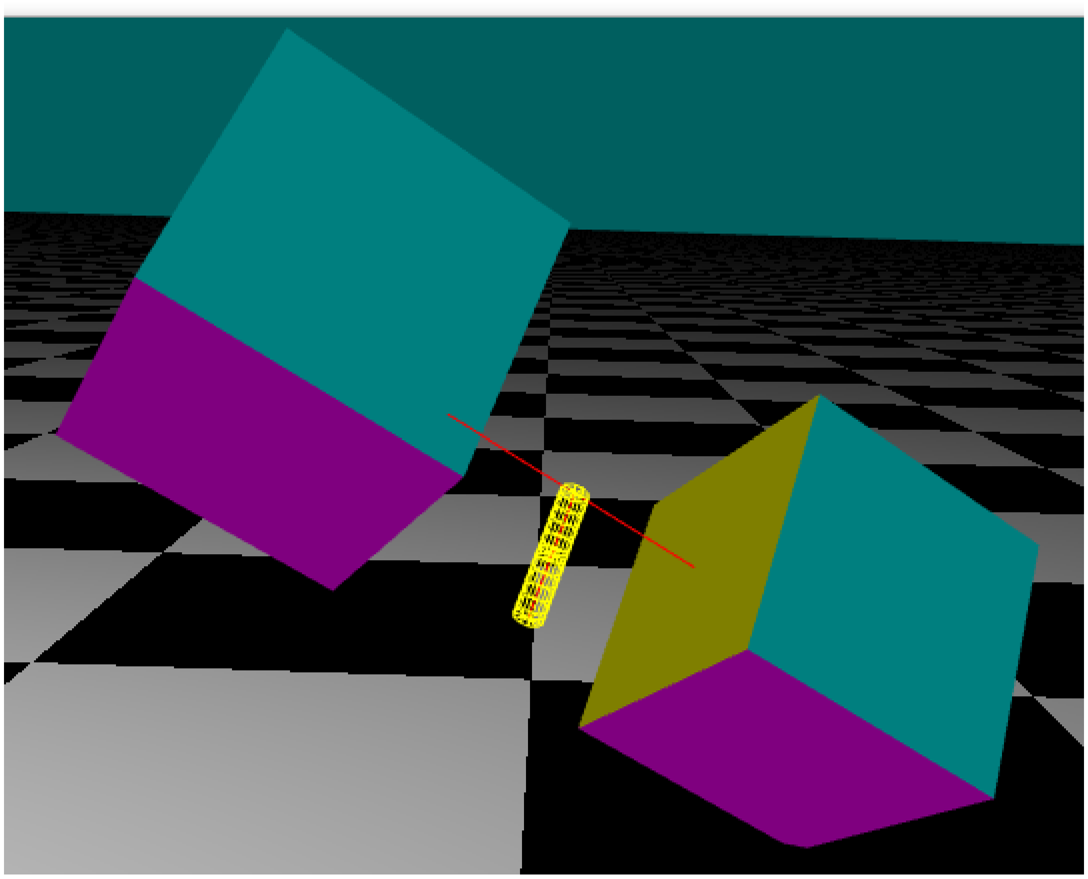
Bone rotation

- When to rotate a bone
  - I rotate a bone if it's highlighted and the user is dragging.
- Rotation axis and rotation speed
  - During a bone rotation, the axis the bone rotates around is defined as the cross product of the camera direction and the mouse direction. It then rotates GUI.rotationSpeed radians around that axis (currently set as .1 in a constant)
- Rotation of a parent bone
  - To do the actual rotation on a single bone, I calculate the rotationQuat from the axis and angle and set the bone’s current rotation quat to the product of the calculated rotationQuat and its current rotation Quat.
  - I then apply the calculated rotationQuat on the bone's tangent and translate it to the bone's position.
  - All its children are rotated as well based on the following logic.
- Rotation of a child bone
  - Child bones are a bit different in the sense that both their position and endpoint are updated
  - We get the child's position by applying the rotation quat on the difference between the child's endpoint and the original parents' position and translating it to its parent's endpoint.

    - For the child's endpoint, rather than translating it by its position, it's translated by its original parent's position, and the axis that the rotation quat is applied on is the difference between the child's endpoint and the original parent's position.
- Rolling of bones with Left and Right Arrow Keys
    - On the left and right arrow key presses, If there's a bone highlighted, I call the same rotate bone function, except that instead of rotating on the previously mentioned axis, we rotate the bone on the bones tangent, and rather than rotating it GUI.rotationSpeed radians, we rotate it GUI.rollspeed radians.
    - The left arrow rotates the bone around the negative tangent, and the right arrow rotates around the positive tangent.
- Results
    - Before rotation:



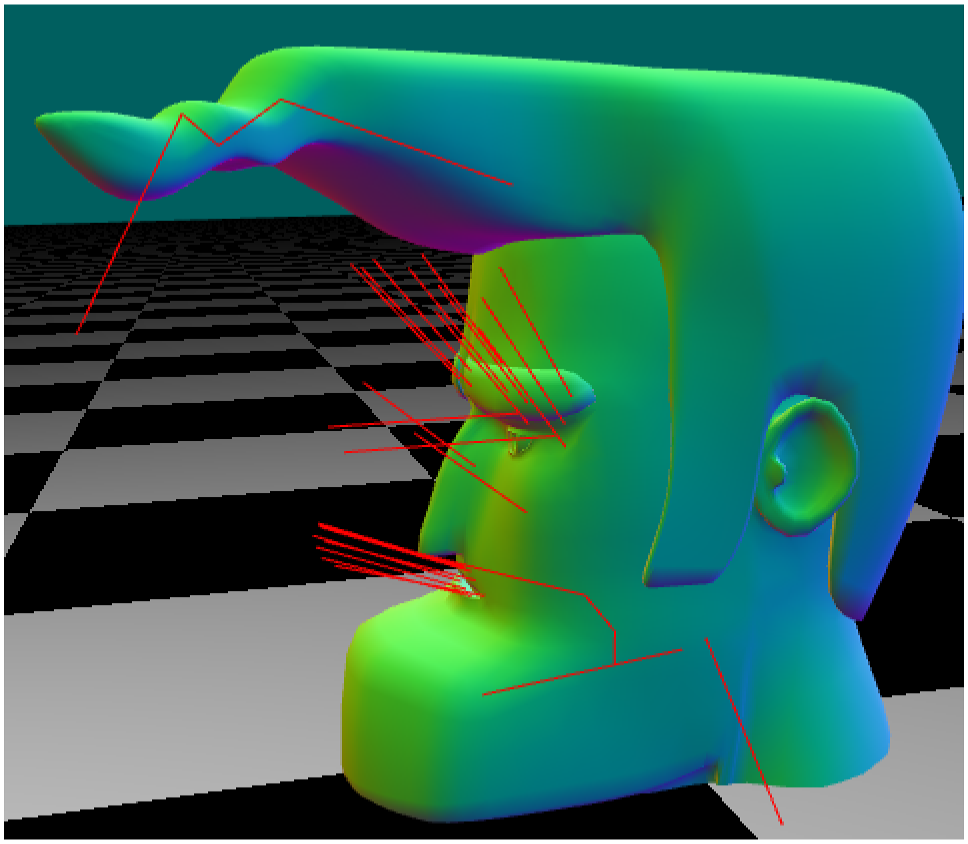
- After rotation:



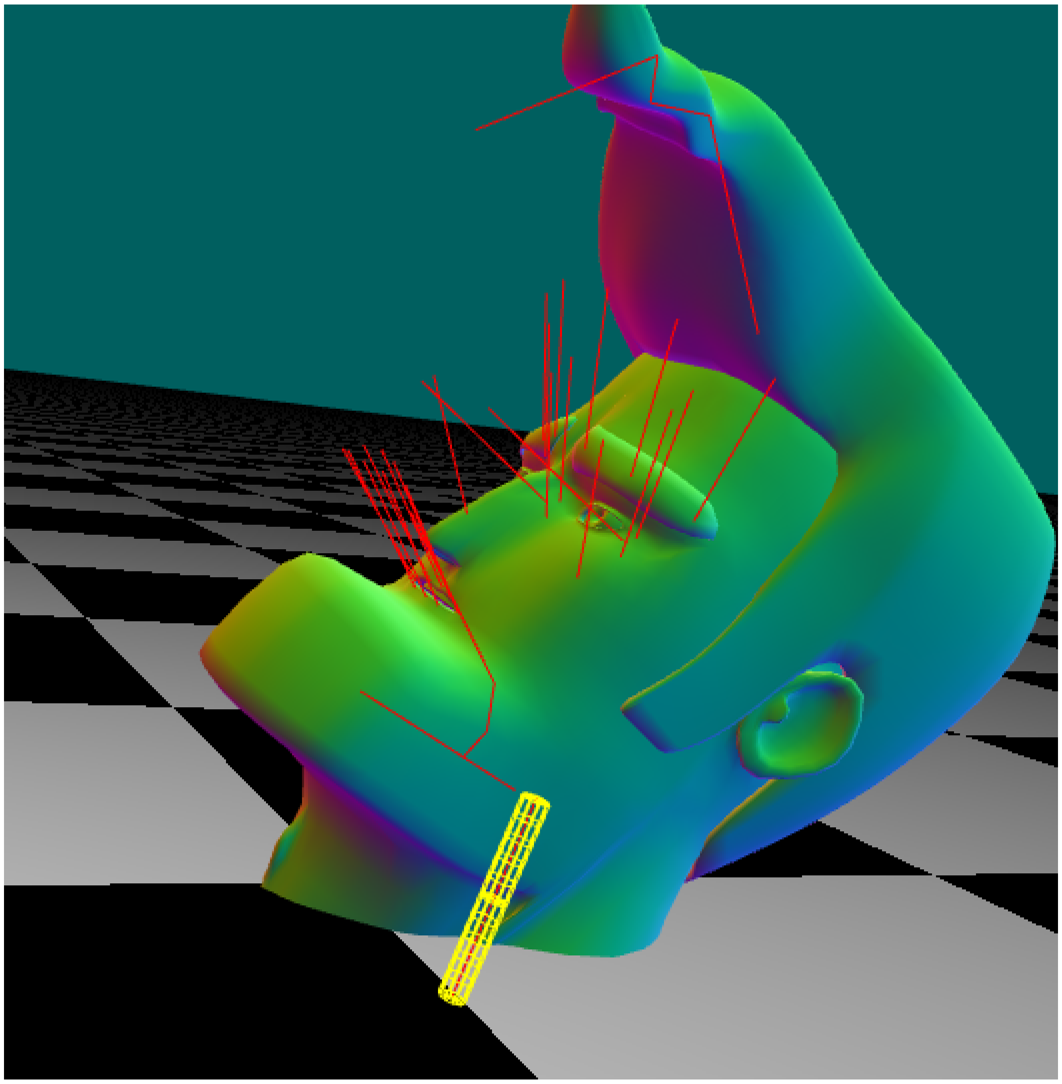
Linear Blend Skinning

- Purpose
  - Gets an object to deform based on the orientation of its bones
- How
  - This is done in the scene shader
  - I calculate the bone translation of v0-4 using jRots and jTrans, which I index into using skin indices
  - I multiply the translation by the weight of the v, which is stored in skin weights.
  - I sum all of these weighted translations and use that as the translation vector
- Results

- Before rotation:



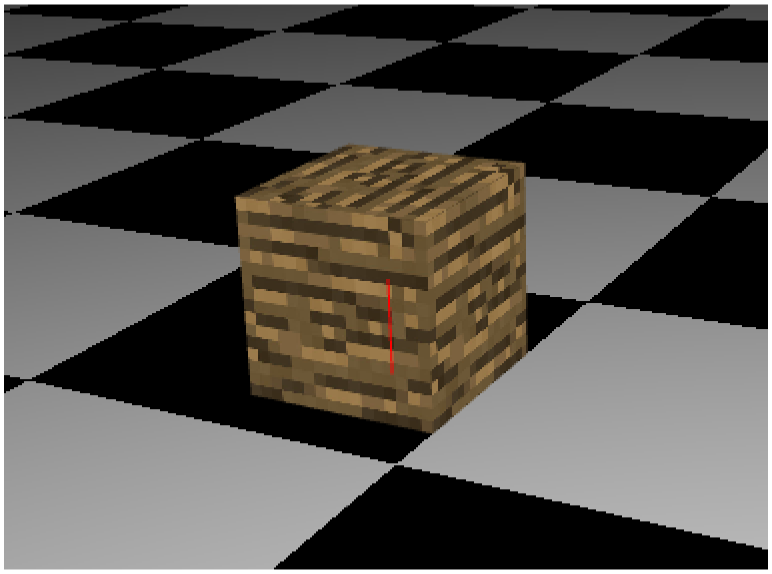
- After rotation:



Texture Mapping

- How
    - I added a texture map to the model render pass if the meshes image source wasnt null
    - I created a new vertex and fragment shader for texture mapping, which are identical to the scene shaders except I define a uniform sampler2D variable called img, and in the fragment shader, I set gl_FragColor = texture2D(img, uv), uv being the texture coordinates passed in from the vertex shader
- Results

○



Major Bug Fixes

- One major bug that took me forever to figure out was when I had gridded cylinders enabled for highlighting, my rotations got progressively slower and slower. The problem ended up being the way I was setting up the gridded cylinder in drawing. I was adding a new attribute to the renderpass before each draw and, rather than updating the attribute, it added a whole new one, making the setup time ridiculously long the more frames that were drawn. So the appearance of slower rotations was actually due to lag, not logic errors in the code. I edited the code of the renderpass to clear out duplicate attributes, and it is extremely faster now.

Final Notes

- Everything works exactly as intended. Had to work through many bugs, especially with the cylinder intersection and gridded cylinder for highlighting, but I finally got it to work in the end. See above if you haven't already for implementation details.