Computer Graphics HW3

110550085 房天越

I. Implementation

1. **main.cpp**
   In main function, I create the shader program with the according vertex shader, geometry shader, and fragment shader. Then I pass the required parameters to the shader, including M, V, P, material, light, shininess, and a dissolve factor for dissolve effect, etc.
   In main.cpp, I also control the parameter for dissolve factor, and switch the current program according to the keyboard input.

2. **L, V, R, and N Vectors**
   For the following shading methods, we need to get L, V, R, and N vectors to simulate the light, the calculation method is as following:
   L = normalize(light.position - worldPos.xyz), which is the vector pointing to the light source.
   V = normalize(cameraPos - worldPos.xyz), which is the vector pointing to the position of camera.
   R = normalize(reflect(-L, normal)), which is the vector of the reflection light.
   N = normalize(normal), which is the normal vector of the point.

3. **Blinn-Phong Shading**
   For Blinn-Phong Shading, I calculate ambient, diffuse, and specular using the Ka, Kd, Ks, La, Ld, Ls defined in main.cpp, with the way like Phong shading method, the only difference is that I use H to replace R, the way to calculate H is H = normalize(L+V).
4. **Gouraud Shading**
   The main part for calculating is in vertex shader for this shading method because interpolation is required. The way to implement this is like Phong Shading.
5. **Flat Shading**
   The most important for this shading method is geometry shader, in which I calculate the normal by taking the cross product of worldPos[2]-worldPos[0] and worldPos[1]-worldPos[0], then pass this normal to

fragment shader for result computation.

6. **Toon Shading**
   The way to do this is relatively simple. I define three color levels, and calculate the intensity by taking the dot product of L and V, if it is less than 0, assign it with dark brown, else if it is larger than 0.8, assign it with bright skin color, else assign it with the normal brown color.
7. **Border Effect**
   The way to do this is to take the dot product of N and V, if it is very close to 0 (-0.2 < x <0.2 in my implementation), then assign it with white color, or assign it with the original texture color.
8. **Dissolve Effect**
   The way to do this requires a specific method "discard", I pass a value called dissolveFactor to fragment shader using uniform, it is initialized to -30, and it increases by 0.3 per frame. If the x position of the vertex is less than the dissolveFactor, discard it. If switch back to other shading methods, reset the value to -30.

II. Problems I Met and Solutions

The main problem that I met is to implement flat shading. I was not familiar with the way to write geometry shader before, and it took me some time to realize how to implement it, including what it means to take max_vertices and some other implementation details.

Another problem is that I did not know how to implement dissolve effect because I did not know "discard", I tried to implement that by setting the vertex color to the background color, but I think that is not what is desired to be seen, so I had a discussion with my classmate and found that I could use discard to implement the effect, and also found that I could use the x coordinate to determine what vertices to be discarded.