CITS3003 Project - 2021 Semester 1

Thomas Cleary - 21704985

Contents

1 Overview of Project

This section provides a summary of the tasks that were and were not successfully implemented in my scene.

It also discusses the way in which I have used commenting in *scene-start.cpp*, *vStart.glsl* and *fStart.glsl* to indicate which section of code I have written relates to which task in the project specification.

1.1 Completed Tasks

I was able to complete tasks A - J(b) successfully.

1.2 Incomplete Tasks

I was unable to complete task J(c).

1.3 Commenting Explained

The comments I have added to the three files listed above use a certain structure to indicate which task the code is related to.

Any code placed between two comments, as shown below, relates to that specific task.

```
// *** Task X
//
... Task X code in here ...
//
// Task X ***
```

Sometimes these comments may be nested within other task's comments.

2 Task Implementation Discussion

This section will discuss implementation details of each task implemented from the project specification. I have omitted most of the comments from the blocked code samples in each task section.

a Task A - Camera Rotation

To implement the camera rotation functionality the display callback function had to be modified. Rotate functions from *mat.h* were used to create a rotation matrix that would then be applied to the view matrix before translation. The exact angles that define the rotation about the x and y axes were provided in the *camRotUpAndOverDeg* and *camRotSidewaysDeg* variables respectively.

```
mat4 rotateXY = RotateX(camRotUpAndOverDeg) *
                RotateY(camRotSidewaysDeg);

view = Translate(0.0, 0.0, -viewDist) *
       rotateXY;
```

b Task B - Object Rotation

To rotate objects in the scene a rotation matrix is defined in much the same way as Task A. This time however, we include a rotational component about the z-axis, and the angles supplied to each rotate function are stored in the sceneObj struct for the object being rotated.

To produce a rotational effect like the one in the example video, I did have to negate the angle supplied to the *RotateX()* function.

```
mat4 rotateXYZ = RotateX(-sceneObj.angles[0]) *
                 RotateY( sceneObj.angles[1]) *
                 RotateZ( sceneObj.angles[2]);

mat4 model = Translate(sceneObj.loc) *
             Scale(sceneObj.scale)   *
             rotateXYZ;
```

To change the scale of the texture on a scene object the *texScale* variable is already passed to the shader program for us. We simply multiply the texture coordinate by this scaling factor in the call to *texture2D()* in the fragment shader.

```
gl_FragColor = color * texture2D( texture, texCoord * texScale );
```

c Task C - Adjusting Ambient/Diffuse/Specular Light and Shine

Two callback functions, *adjustAmbientDiffuse(vec2 ambDiff)* and *adjustSpecularShine(vec2 specshine)* were created to adjust the ambient, diffuse, specular and shine values of a scene object.

```
static void adjustAmbientDiffuse(vec2 ambDiff) {
    sceneObjs[toolObj].ambient += ambDiff[0];
    sceneObjs[toolObj].diffuse += ambDiff[1];
}


static void adjustSpecularShine(vec2 specShine) {
    sceneObjs[toolObj].specular += specShine[0];
    sceneObjs[toolObj].shine    += specShine[1];
}
```

These functions were modelled on the ones provided in the skeleton code such as *static void adjustRedGreen(vec2 rg)* which adjust the color of a light depending on the movement of the mouse and button held.

These call back functions are then associated with the correct menu entry.

```
int AMB_DIF_SPEC_SHN_MENU_ID = 20;

else if (id == AMB_DIF_SPEC_SHN_MENU_ID) {
    setToolCallbacks(adjustAmbientDiffuse, ident_2,
                     adjustSpecularShine,  ident_2);
```

d Task D - Allow Closer Object Views

To stop objects from clipping so early when we try to get closer to them with the camera, the *nearDist* variable in the *reshape()* callback function can be reduced.

```
// Adjusted from 0.2 -> 0.01
GLfloat nearDist = 0.01;
```

Changing the value to 0.01 allows the camera to get much closer to scene objects without the objects triangles prematurely clipping.

e Task E - Window Reshaping

To allow all objects that are visible in a square window, to still be visible when the window width and height are adjusted further changes to the the *reshape()* callback function are needed.

```
GLfloat left, right, bottom, top;

GLfloat near = nearDist;
GLfloat far = 100.0; // As defined in skeleton code

if (width < height) {
    left   = -nearDist;
    right  = -left;
    bottom = -nearDist * (float)height / (float)width;
    top    = -bottom;

}
else { // originally what the skeleton code provided
    left   = -nearDist * (float)width / (float)height;
    right  = -left;
    bottom = -nearDist;
    top    =  nearDist;
}

projection = Frustum(left, right, bottom, top, near, far);
```

The values assigned to variables after the *else* clause (when window width is greater than it's height) are essentially what the skeleton code provided. To assign the values if the window width is less than its height (the *if* clause):

- left and right become what were bottom and top in the *else* clause

- bottom and top become what were left and right in the *else* clause except we change the order of operations to multiply *-nearDist* by *height* first and then divide by *width*.

Figure 1: Square Window

Figure 2: Width > Height Window

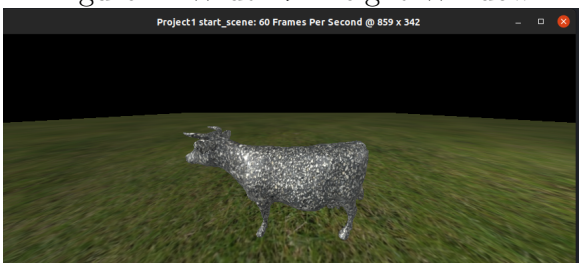

Figure 3: Height > Width Window

f Task F - Light Reducing with Distance

To simulate light reducing with distance a *lightIntensity* scalar was computed to the scale the *(ambient + diffuse + specular)* light for a vertex.

We define this scalar as 1 divided by the distance from the vertex to the light source. As the distance from the lightSource to the vertex increases this scalar gets closer to zero. As it decreases the scalar approaches infinity (but will not as we use *NON_ZERO_VALUE* to avoid a division by zero error).

This scalar then modifies the value of *(ambient + diffuse + specular)* for a vertex depending on how far away it is from the light source. (further away = smaller, closer = larger)

```
float NON_ZERO_VALUE = 0.01; // to avoid division by zero
float vertexToLightDist = length(Lvec);

float lightIntensity = 1.0 / (NON_ZERO_VALUE + vertexToLightDist);

color.rgb = globalAmbient +
          ( (ambient + diffuse + specular) * lightIntensity );
```

g Task G - Move Lighting to Fragment Shader

The lighting calculations in the vertex shader were not removed but commented out, to show what was done in Task F.

However, to move the lighting calculations into the fragment shader, several variables were now required to be shared between the vertex and fragment shader.

Vertex shader after Task G:

```
attribute vec3 vPosition;
attribute vec3 vNormal;
attribute vec2 vTexCoord;

varying vec4 position;
varying vec3 normal;

varying vec2 texCoord;

uniform mat4 ModelView;
uniform mat4 Projection;

void main()
{
    position = vec4(vPosition, 1.0);
    normal = vNormal;
    texCoord = vTexCoord;

    gl_Position = Projection * ModelView * position;
}
```

position and *normal* are now varying variables shared to the fragment shader to carry out the lighting calculations. Hence, *varying vec4 color* could be removed as this will be calculated solely in the fragment shader now.

The same calculations are copied over to the fragment shader, hence we require these variables in there now. Most of which are just copied over from the vertex shader.

Variables now passed to fragment shader:

```
varying vec4 position;
varying vec3 normal;
varying vec2 texCoord;

vec4 color;

uniform vec3 AmbientProduct, DiffuseProduct, SpecularProduct;
```

```
uniform mat4 ModelView, Projection;
uniform float Shininess;

uniform vec4 LightPosition1;
```

I will not show the lighting code as this has simply been transferred over from the vertex shader.

h Task H - Get Specular Highlights to Shine Towards White

To get specular highlights to shine towards white I needed to seperate the specular light component from the fragments color calculation in the fragment shader. To ensure that the specular light is independant of the light's color.

```
color.rgb = globalAmbient +
            ((ambient1 + diffuse1) * lightIntensity)

gl_FragColor = color *
               texture2D( texture, texCoord * texScale ) +
               vec4(specular1 * lightIntensity, 1.0);
```

i Task I - Adding a Second Light

Adding a second light to the scene is mostly similar to the one implemented already in the skeleton code. However as this light is directional it is not affected by the translation of the camera.

(The non directional light source is altered by the view matrix where as this new light source is simply altered by the rotation matrix used to define the view matrix)

Task I in *scene-start.cpp*

```
(inside the init() function)

int LIGHT_ID = 55;

// Create Second Light same as first but with negative x coord
addObject(LIGHT_ID);
sceneObjs[2].loc = vec4(-2.0, 1.0, 1.0, 1.0);
// Make slightly bigger to differentiate from light 1
sceneObjs[2].scale = 0.2;
sceneObjs[2].texId = 0;
sceneObjs[2].brightness = 0.2;

...
...
...

(inside the display() function)
// Second light position to be affected only by camera rotation.
SceneObject lightObj2 = sceneObjs[2];
vec4 lightPosition2 = rotateXY * lightObj2.loc;
```

We also now needed to alter the lighting calculations in the fragment shader to account for this new light source. Light reduction does not apply to a directional light source so it is not affected by *lightIntensity*

```
gl_FragColor = color *
               texture2D( texture, texCoord * texScale ) +
               vec4(specular1 * lightIntensity + specular2, 1.0);
```

j Task J(a)(b) - Object Deletion and Duplication

To be able to delete or duplicate a scene object we first need to be able to select the object that we wish to delete or duplicate. A menu is created to select an object that is currently in the scene (excluding what I have called permanent objects which include the ground and 2 light objects)

```
int selectActiveObjectMenuId = glutCreateMenu(selectActiveObjectMenu);

int OBJ_NAME_LEN = 128; // same as gnatidread

for (int i = NUM_PERMANENT_OBJS; i < nObjects; i++) {
  char objName[OBJ_NAME_LEN];

  if (sceneObjs[i].meshId != NULL) {
    // ensure unique id for menu entry
    int objectID = 100 + i;

    // Create Unique name for object in menu
    sprintf(objName, "%d", objectID-100);
    strcat(objName, " - ");
    strcat(objName, objectMenuEntries[sceneObjs[i].meshId - 1]);
    if (currObject == i) {
        strcat(objName, " - ACTIVE");
    }

    glutAddMenuEntry(objName, objectID);
  }
}
```

To then delete or duplicate (or 'clone' as I have called it in the program) a scene object we need two menu entries to select these options.

```
// Show menus related to sceneObjects only when one is selected
if (currObject != -1) {
    if (currObject > NUM_PERMANENT_OBJS - 1) {
        // don't want to delete/clone ground or lights
        glutAddMenuEntry("Delete object", 51);
        glutAddMenuEntry("Clone Object", 52);
    }
```

Below are the functions that handle the scene object deletion and duplication. Each cloned object is spawned slightly in the positive X direction of it's parent object.

```
static void deleteObject(int objID) {
    sceneObjs[objID].meshId = NULL;
    currObject = -1;
    deletedObjects++;
    makeMenu(); // Update menu to hide from the object list
}


static void cloneObject(int objID) {
    if (nObjects < maxObjects) {
        // nObjects is last item added index + 1
        sceneObjs[nObjects] = sceneObjs[objID];

        sceneObjs[nObjects].loc.x += 0.75;

        currObject = nObjects;
        // nObjects is correct index for currObject (1 behind)
        toolObj = currObject;
        nObjects++;

        // from addObject skeleton code
        setToolCallbacks(adjustLocXZ, camRotZ(),
                         adjustScaleY, mat2(0.05, 0, 0, 10.0));
        glutPostRedisplay();

        makeMenu(); // update menu to add obj to list in menu
    }
}
```

k Task J(c) - Adding a Spotlight

As mentioned previously, I was unable to complete this task.