# CITS3003 – Project Report

Reiden Rufin – 22986337
Nathan Eden – 22960674

## Opening Remarks

We initially had problems in terms of project compilation such as missing X11 XInput.h. To combat this we had to install a package.

sudo apt-get install libxi-dev

Some dependencies also had to be installed to make the project run:

sudo apt install cmake libxmu-dev g++ libx11-dev libgl1-mesa-dev libglu1-mesa-dev xorg-dev libxi-dev

***Tasks marked with a green tick indicate that we have managed to implement that specific task***

*Note: Tasks A - F uses the "fStartOld.glsl" and "vStartOld.glsl". Rest of the other tasks utilizes the "fStart.glsl" and "vStart.glsl"*

## Task A – ✅

As we read the Task description of camera rotation based on mouse inputs, it became clear to us that we had to modify something inside the display callback.

We inferred that it had something to do with the in-built functions RotateX() and RotateY() as the task asked us to rotate both horizontally and vertically.
Similarly, looking at the global variables given to us, we noticed that

- camRotSideWaysDeg was commented with "rotates the camera around the center"
- camRotUpandOverDeg was commented with "rotates the camera up and over the center

These variables directly corresponded to the task asking us to rotate the scene that is vertical to the ground plane and change the elevation angle
To simply change the view we just multiply these two variables and multiply it to the translation view.

```
mat4 rotate_cam_x = RotateX(camRotUpAndOverDeg);
mat4 rotate_cam_y = RotateY(camRotSidewaysDeg);
mat4 rotate = rotate_cam_x * rotate_cam_y;
view = Translate(0.0, 0.0, -viewDist) * rotate;
```

## Task B – ✅

Task was fairly straightforward, as it is identical to Task A, but this time we are asked to rotate the object itself. To do this we decided to use the angles array via .angles[] and then rotate respectively with the already existing RotateX() RotateY() and RotateZ() along each axis. This lets us create a rotation matrix.

One thing to point out is the way in the demo video, it moves inverted from the mouse. To match the demo, we also had to invert our RotateX() and RotateZ();
One problem we did run into was if we multiplied it such that (x * y * z) it would not rotate the object based on the object's position, rather it rotated it based on the camera position. The fix to this was just inverse the rotation such that (z * y * x).

```
mat4 rotate = RotateZ(sceneObj.angles[2]) * RotateY(sceneObj.angles[1])
* RotateX(-sceneObj.angles[0]);
mat4 model = Translate(sceneObj.loc) * Scale(sceneObj.scale) * rotate;
```

We additionally pass in a texScale to the fragment shader where we would replace the 2.0 to a uniform variable texScale to let it scale in conjunction with object rotation.

```
uniform float texScale; //TASK B

void main()
{
   gl_FragColor = color * texture2D( texture, texCoord * texScale);
}
```

## Task C – ✅

Similar to the functions already in the skeleton coded (in regards to adjusting brightness) we added two more functions

- One for ambient diffuse
- One for specular shine

We then call these functions in the materialMenu() with an id of 20 (found in makeMenu()).

We also call the setToolCallbacks in this menu as it takes 4 parameters, which consists of calling two more functions with their identity matrix in between. As such, we make adjust_ambient_diff act in respect to the left click and adjust_spec_shine respective to the middle mouse button.

```
static void adjust_ambient_diff(vec2 ad)
{
   sceneObjs[toolObj].ambient += ad[0];
   sceneObjs[toolObj].diffuse += ad[1];
}
static void adjust_spec_shine(vec2 ss)
{
   sceneObjs[toolObj].specular += ss[0];
   sceneObjs[toolObj].shine += ss[1];
}
```

```
else if (id == 20)
   {
       toolObj = currObject;
       setToolCallbacks(adjust_ambient_diff, mat2(1, 0, 0, 1),
                        adjust_spec_shine, mat2(5, 0, 0, 5));  //ENDOF
TASK C
   }
```

## Task D – ✅

We found out that the variable nearDist is what determined the distance the triangles clipped from the camera. We simply set it to 99% smaller going from the original value of 0.2 to 0.002 as sort of an “Extreme” to see the differences and it works. By setting it smaller it adjusts the viewing volume so the camera can move closer to the object with less clipping to ensue.

```
GLfloat nearDist = 0.002; //adjusted for TASK D
```

## Task E – ✅

For this task we are asked to keep the objects visible whenever window is resized
To do this we can simply check whenever the height of the window is less than the width and use the already written Frustum() projection existing in the template
But if the width is now less than the height we have to reverse the division of width / height to height / width.
Along with some playing around with the parameters inside the Frustum() for the else statement, we got it to work (trial and error a lot.)

On a side note, we found in the solutions to lab 5 in regards to the question about resizing underneath their reshape() func was similar.

```
//TASK E
   if ( height < width )
   {
       projection = Frustum(-nearDist * (float) width / (float) height,
                             nearDist * (float) width / (float) height,
                            -nearDist, nearDist,  nearDist, 100.0);
   }
   else
   {
```

```
        projection = Frustum(-nearDist, nearDist,
                             -nearDist * (float) height / (float) width,
                             nearDist * (float) height / (float) width,
                             nearDist, 100.0);
    }
    //ENDOF TASK E

                             nearDist, 100.0);
     }
```

## Task F – ✅

With a quick ctrl + f to find the destination folder for the vertex shader, we immediately knew that we had to work around vec3 Lvec as it was the vector to the light
To get the light we had to pass in the attenuation formula.
We chose to have to light reduce from 0.1 + the length of the vector as it yielded good results
This allowed the lighting to be dependent on the distance from the light source

```
float light_distance = 0.1 + length(Lvec);
float light = 1.0/(1.0 + 1.0*length(Lvec) + light_distance * light_distance);
```

The equation for light is calculated from the attenuation formula given in Lecture 14.

- Hence, the light attenuation is generally modelled by a quadratic term $\frac{1}{a+bd+cd^2}$ (where $a, b, c$ are user defined parameters)

[1]

[1] Taken from CITS3003 Lecture 14 regarding shading, slide 17.

## Task G – ✅

Moving the variables and functions from the vertex shader was quite a hassle as we had to change the types of some variables from attributes to varying. Particularly, the variables *position, normal and texCoord.* The lighting calculations also had to be moved, but not much change was needed for this part.
This allowed us to change the lighting from each vertex to lighting per fragment

```
//TASK G - moving variables to fragment shader
uniform vec3 AmbientProduct, DiffuseProduct, SpecularProduct;
uniform mat4 ModelView;
uniform float Shininess;
uniform vec4 LightPosition;
uniform float LightBrightness;
uniform vec3 LightColor;
```

## Task H – ✅

To achieve a specular highlight that steers towards white we had to extract the LightBrightness and LightColor in the .cpp file and access them inside the fragment shader

```
glUniform4fv(glGetUniformLocation(shaderProgram, "LightPosition"),
                1, lightPosition);
   CheckError();
glUniform1f(glGetUniformLocation(shaderProgram, "LightBrightness"),
               lightObj1.brightness);
   CheckError();
glUniform3fv(glGetUniformLocation(shaderProgram, "LightColor"), 1,
lightObj1.rgb);
```

```
  CheckError();
```

We then multiplied this to the already existing specular light and the Specular product. This resulted in the specular shine always being white rather than the light's color.

```
vec3 ambient = (LightColor * LightBrightness) + AmbientProduct;
vec3 diffuse = Kd * (LightColor * LightBrightness) * DiffuseProduct;
vec3 brightness = vec3(2,2,2);
vec3 specular = Ks * (SpecularProduct + brightness);
...
color.rgb = globalAmbient + ((ambient + diffuse + specular) * light)
```

A good source for this part is as shown here.[2]The calculation for the final rgb is as specified (ambient + diffuse + specular) multiplied by our light calculations.

## Task I – ✅

To create a second light we would first have to modify the display function, implementing in a way that is similar to the first light. The only difference here is that because the second light is directional we would have to multiply the camera rotation by the light location instead of the view

```
SceneObject lightObj2 = sceneObjs[2];
vec4 lightPosition2 = rotate * lightObj2.loc; //directional so multiply
the camera rotation by the light location
```

Similarly we would load the uniform variables into the fragment shader for light 2

```
glUniform4fv(glGetUniformLocation(shaderProgram, "LightPosition2"),
                1, lightPosition2);
   CheckError();
   glUniform1f(glGetUniformLocation(shaderProgram, "LightBrightness2"),
               lightObj2.brightness);
   CheckError();
   glUniform3fv(glGetUniformLocation(shaderProgram, "LightColor2"), 1,
```

[2] https://learnopengl.com/Lighting/Basic-Lighting

```
lightObj2.rgb);
   CheckError();
```

Inside the fragment shader we pass on these variables

```
uniform vec4 LightPosition2;
uniform float LightBrightness2;
uniform vec3 LightColor2;
```

Additionally we would also have to add the properties of the second light inside the init() function

```
//TASK I
   addObject(55); // Sphere for  light 2
   sceneObjs[2].loc = vec4(3.0, 1.0, 1.0, 1.0);
   sceneObjs[2].scale = 0.1;
   sceneObjs[2].texId = 0; // Plain texture
   sceneObjs[2].brightness = 0.2; // The light's brightness is 5 times
this (below).
 // ENDOF TASK I
```

One issue we had was that the light wasn't as bright as the demo video. To combat this we statically adjusted the scale and brightness inside the .cpp file just under the declaration of second light creation, although we think this is not necessary as it functionally performs the way we intended it to.

```
lightObj1.scale = 0.5;
lightObj1.brightness = 1.0;
```

## Task J – ✅

The first part of this task required us to add an option to the menu to delete/duplicate objects. Our assumptions were that it referred to the most recently created object.
To start we added the Menu Entry inside the makeMenu() function with an id of 97 and 98

```
glutAddMenuEntry("Delete Object", 97);
glutAddMenuEntry("Duplicate Object", 98);
```

The Delete Function just simply checked if we do have an object in the scene, if so then we cannot delete the object.
We then loop through to find the id of the most recently created object then set it to the current object
Decrement the total number of objects then finally redraw the scene

```
//TASK J - DELETE OBJECT
static void delete_object(int object)
{
   if (object < 0 || object ⩾ nObjects) //check if we do have an object in scene
   {
       printf("Could not delete the object. Maybe something went wrong?\n");
   }
   for (int i = object; i < nObjects - 1; i++)
   {
       sceneObjs[i] = sceneObjs[currObject];
   }
   nObjects--;
   glutPostRedisplay(); //redraw scene
}
```

The Duplicate Function checked if we have exceeded the total number of maximum objects allowed in the program, if so then simply return and print a message that it has been exceeded
We would then copy the object and set both the tool and the current object to the new object. Redraw the scene

```
// TASK J - DUPLICATE OBJECT
static void duplicate_object(int object)
{
   if(nObjects == maxObjects) //check if we have exceeded the maximum
amount of objects
   {
       printf("Exceeded the maximum amount of objects\n");
       return;
   }
   sceneObjs[nObjects] = sceneObjs[object]; //copies the object
   toolObj = currObject = nObjects++; //sets both the tool and the
current object to the new object, increment total number of objects
   setToolCallbacks(adjustLocXZ, camRotZ(),
                    adjustBrightnessY, mat2(1.0, 0.0, 0.0, 10.0));

   glutPostRedisplay(); //redraw scene
}
```

Finally, we would need to link these into the main menu by passing on their id then calling their respective functions

```
//TASK J
   if (id == 98 && currObject >= 0)
   {
       duplicate_object(currObject);
   }
   if(id == 97 && currObject >= 0)
```

```
    {
        delete_object(currObject);
    }
```

The harder part of Task J and probably the hardest part of the project was implementing the spotlight. To do this, we decided to add a third light.
Similarly to the first two lights, we would create the second light in the display then pass on uniform variables for its Position, Brightness, Color with the addition of this light's location so that they could be independent of the first and second light. We also found out that it wasn't as bright as in the screenshot provided in David's Guide, so we manually increase the brightness and scale just like we had done for light 2.

```
//TASK J - SPOTLIGHT
    SceneObject lightObj3 = sceneObjs[3];
    vec4 lightPosition3 = view * lightObj3.loc;
    float light_lateral = lightObj3.angles[1];
    float light_vertical = lightObj3.angles[2];

    glUniform1f(glGetUniformLocation(shaderProgram, "lateral"),
light_lateral);
    CheckError();
    glUniform1f(glGetUniformLocation(shaderProgram, "vertical"),
light_vertical);
    CheckError();
    glUniform4fv(glGetUniformLocation(shaderProgram, "LightPosition3"),
                 1, lightPosition3);
    CheckError();
    glUniform1f(glGetUniformLocation(shaderProgram, "LightBrightness3"),
                 lightObj3.brightness);
    glUniform3fv(glGetUniformLocation(shaderProgram, "LightColor3"), 1,
```

```
lightObj3.rgb);
   CheckError();
```

We also add the new light to the init() function by passing on sceneObjs[3].

```
//TASK J
   addObject(55); // Sphere for  light 3
   sceneObjs[3].loc = vec4(2.0, 3.0, 1.0, 1.0);
   sceneObjs[3].scale = 0.1;
   sceneObjs[3].texId = 0; // Plain texture
   sceneObjs[3].brightness = 0.2; // The light's brightness is 5 times
this (below).
```

As this third light is not already built within the menu, we would have to add it using glutAddMenuEntry with a value of 90 so it would be below our Move Light 2 menu buttons.

```
glutAddMenuEntry("Move Light 3", 90);
glutAddMenuEntry("Change Direction", 91);
```

David's guide also suggested to implement a change direction submenu, which would prove useful for the final part.
Then in the lightMenu similar to the first two lights, but with the addition of Change Direction giving it the value 91 so that it would be directly below out Move Light 3 button:

```
else if (id == 90) //TASK J SPOTLIGHT
   {
       toolObj = 3;
       setToolCallbacks(adjustLocXZ, camRotZ(),
                        adjustBrightnessY, mat2(1.0, 0.0, 0.0, 10.0));
   }
   else if (id == 91){
       toolObj = 3;
```

```
        setToolCallbacks(RotateObj, camRotZ(), adjustBrightnessY,
mat2(1.0, 0.0, 0.0, 10.0));
    }
```

Inside of scenestart.cpp we had to create a function RotateObj() which was necessary for the callback function associated with the ‘change direction’ entry in the menu for light 3. This is seen above as the first parameter of setToolCallbacks for id == 91.

```
static void RotateObj(vec2 xz){
    sceneObjs[toolObj].angles[2]+=-20*xz[0];
    sceneObjs[toolObj].angles[1] +=-20*xz[1];
}
...
else if (id == 91){
        toolObj = 3;
        setToolCallbacks(RotateObj, camRotZ(), adjustBrightnessY,
mat2(1.0, 0.0, 0.0, 10.0));
    }
```

Inside the fragment shader we would have to pass on the uniform variables from earlier, similarly to the first two lights again. We also add lateral and vertical which will be used for the spotlight’s direction

```
vec4 colour_3;
...
//TASK J - SPOTLIGHT
uniform vec4 LightPosition3;
uniform float LightBrightness3;
uniform vec3 LightColor3;
uniform float lateral;
uniform float vertical;
```

We created direct so that we could use it to do some of the calculations required for changing the direction of light 3. At this stage it is a vector that points FROM the light source.

David's guide also helped in this scenario for the cutoff calculation, directing us to a site about light casters in OpenGL. See Here.[3]

```
//used for chaning direction of light 3
vec3 direct;
direct.x = cos(radians(vertical))*cos(radians(lateral));
direct.y = sin(radians(lateral));
direct.z = sin(radians(vertical))*cos(radians(lateral));
```

Here we declared theta which incorporated direct from earlier albeit we normalized and inverted it

```
float theta = dot(L3, normalize(-direct));
```

Theta is used to define the cutoff point for the spotlight, so that anything within the spotlight will be illuminated and everything outside of the spotlight will only have ambient lighting. Here it is the dot product between the direction vector which is inverted towards the light source, and the direction the spotlight is aiming at. This results in a very clear cut off point between the spotlights cone shaped light and everything outside it. The value of theta was used to determine the values of color3 and specular3.

```
float light_distance_3 = 0.1 + length(Lvec3);
   if (theta > 0.9){
       colour_3 = vec4(ambient3 + light_distance_3 * (diffuse3), 1.0);
   }
   else{
       colour_3 = vec4(ambient3, 1.0);
   }
```

Next we add color3 to gl_Fragcolor

```
gl_FragColor = (color + colour_3) * texture2D(texture, texCoord *
texScale);
```

[3] https://learnopengl.com/Lighting/Light-casters

# References:

https://learnopengl.com/Lighting/Basic-Lighting
- Used for Task H Specular Calculation

https://learnopengl.com/Lighting/Light-casters
- Used for Task J Spotlight Cutoff Calculations