CMP203 Coursework Report

Student Name: Joseph Roper

1901881

Git Username: 1901881

Introduction

For my project I have decided to do a scene from one of my favourite games Minecraft. The scene contains a wooden café with tables and chairs outside on a cobble road surrounded by grass.

Controls

W - move camera forward

A – move camera left

S - move camera back

D – move camera right

E - move camera up

Q - move camera down

G - wireframe mode on and off

H - lights on and off

F – open and shut doors

Mouse movement controls cameras rotation

Scene explanation

Texture Class

In the project I have made a texture class which has a constructor function that loads all the textures I use in my scene. Those textures are then saved as numbers in the GLuint variable which is a part of the open gl framework. This means that if I create a texture object of the Texture class in my scene, I can then connect that object to other Texture class objects created in the shape class and pass the object through the shape classes constructor parameter. Meaning that when I create a shape object in the scene if I create an instance of the class which loads the constructor I can input which texture I want the object to have in the parameter as the number the texture corresponds to in the GLuint database.

```
//cafe
Shape sideWalls1 = Shape(11);
Shape sideWalls2 = Shape(11);
```

I decided to have most to nearly all my scene use textures from the game Minecraft as they are readily available match each other nicely as they are from the same game and tile as well as wrap nicely.

Cube

To create the cube, I made 3 arrays the first was for the vertices of the cube the second for the texture coordinates of the cube and the third for the normal of the cube. I then use opengl functions to point towards the arrays I've made and draw them. This cube function shows that I can correctly texture a cube using a t map and use vertex arrays.

```
extern float normsCube24[] = {
    //Front
    0, 0, 1,
    0, 0, 1,
    0, 0, 1,
    0, 0, 1,
    //Back
    0, 0, -1,
    0, 0, -1,
    0, 0, -1,
    0, 0, -1,
    //Left
```

```
extern float texcoordsCube24[] = {
    //Front
    0.25, 0.25,
    0.25, 0.5,
    0.5, 0.5,
    0.5, 0.25,
    //Back
    1, 0.5,
    1, 0.25,
    0.75, 0.25,
    0.75, 0.5,
```

```
extern float vertsCube24[] = {
    //Front
    -1.0,  0.5, 1.0,    // Vertex #7
    -1.0, -0.5, 1.0,    // Vertex #4
    1.0, -0.5, 1.0,     // Vertex #5
    1.0,  0.5, 1.0,     // Vertex #6

    //Back
    -1.0, -0.5, -1.0,   // Vertex #0
    -1.0,  0.5, -1.0,   // Vertex #3
    1.0,  0.5, -1.0,    // Vertex #2
    1.0, -0.5, -1.0,    // Vertex #1
```

Pavement Slabs

For the pavement slabs I made a function that procedurally generates the vertex arrays based on the parameter set when the function is run.

To make it I first created 3 vectors of type GLfloat named; slabVerts, slabNorms and slabTex. I will input the vertex, normal and texture coordinate data into each of these vectors. For this function you can only control how many slabs in the x axis will be made meaning only the x value changes with each loop. In the function I make 4 variables of type double. Y which equals 0.5 which is used to calculate the height of the slab. Z which equals 1 which is used the calculate the depth of the slab. X and scale which will be used to calculate the width of the slabs. I then create a for loop which repeats for the amount set in the function parameter. Within the for loop I push back data into the 3 vectors I made.

```
x = x + 2;
//////////////////Front//////////////////
//Verts
slabVerts.push_back(x); slabVerts.push_back(y); slabVerts.push_back(z);
slabVerts.push_back(x); slabVerts.push_back(-y); slabVerts.push_back(z);
slabVerts.push_back(x+scale); slabVerts.push_back(-y); slabVerts.push_back(z);
slabVerts.push_back(x+scale); slabVerts.push_back(y); slabVerts.push_back(z);
//Norms
slabNorms.push_back(0); slabNorms.push_back(0); slabNorms.push_back(1);
slabNorms.push_back(0); slabNorms.push_back(0); slabNorms.push_back(1);
slabNorms.push_back(0); slabNorms.push_back(0); slabNorms.push_back(1);
slabNorms.push_back(0); slabNorms.push_back(0); slabNorms.push_back(1);
//TexCoords
slabTex.push_back(0); slabTex.push_back(0);
slabTex.push_back(0); slabTex.push_back(1);
slabTex.push_back(1); slabTex.push_back(1);
slabTex.push_back(1); slabTex.push_back(0);
```

As you can see from the picture this is how I create the front face of the slab I calculate the vertexes similar to the vertex array I made for the cube although for the positive X I add the scale and which each loop I increase x with an increment of 2 so that when looped another cuboid is made and placed on the right side of the previous cuboid. The normal for the cuboid were easy to calculate as they are the exact same as the cubes normal data. However, the texture coordinates are calculated to put a simple face texture on the slab compared to the t map used for the cube. I have similar code for the rest of the faces with the data changed accordingly.

```
void Shape::slabRender()
{
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_NORMAL_ARRAY);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);

    glVertexPointer(3, GL_FLOAT, 0, slabVerts.data());
    glNormalPointer(GL_FLOAT, 0, slabNorms.data());
    glTexCoordPointer(2, GL_FLOAT, 0, slabTex.data());

    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glDrawArrays(GL_QUADS, 0, slabVerts.size()/3);
    glDisable(GL_TEXTURE_2D);

    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_NORMAL_ARRAY);
    glDisableClientState(GL_TEXTURE_COORD_ARRAY);
}
```

To render the pavement slabs, I created a separate function so that when the function is run in the scene the slabs aren't being created every frame which would result in a low framerate when running causing lag. In this function I use open gl functions to point towards the data I created in the previous code. Then I enable the texture set it and wrap the texture around the cuboid so that it looks like a pavement stone. I then draw the cuboid using the vertex vectors size data however I divide it by three so that it is read properly as coordinates.

Plane

To make the plane I used similar code to the slab, you input the width and depth of the plane wanted into the function parameters. That data is then used to loop the create face code for how many in the x and z direction with the x and z variables being incremented up by one for every loop.

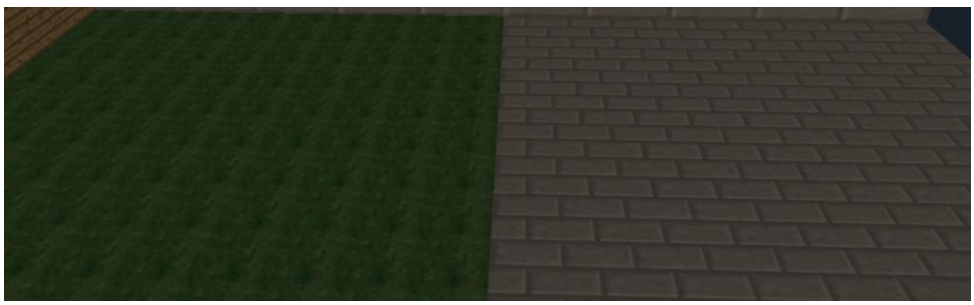
The plane is then rendered the same as the slab just using the data created by the plane create function.

```
double y = -0.5;
double scale = 1;

for (int x = 0; x < Width; x++)
{
    for (int z = 0; z < Depth; z++)
    {
        //Verts
        planeVerts.push_back(x); planeVerts.push_back(y); planeVerts.push_back(z);
        planeVerts.push_back(x); planeVerts.push_back(y); planeVerts.push_back(z + scale);
        planeVerts.push_back(x + scale); planeVerts.push_back(y); planeVerts.push_back(z + scale);
        planeVerts.push_back(x + scale); planeVerts.push_back(y); planeVerts.push_back(z);
        //Norms
        planeNorms.push_back(0); planeNorms.push_back(1); planeNorms.push_back(0);
        planeNorms.push_back(0); planeNorms.push_back(1); planeNorms.push_back(0);
        planeNorms.push_back(0); planeNorms.push_back(1); planeNorms.push_back(0);
        planeNorms.push_back(0); planeNorms.push_back(1); planeNorms.push_back(0);
        //TexCoords
        planeTex.push_back(0); planeTex.push_back(0);
        planeTex.push_back(0); planeTex.push_back(1);
        planeTex.push_back(1); planeTex.push_back(1);
        planeTex.push_back(1); planeTex.push_back(0);
    }
}
```

In the scene I have correctly lit and textured the multiple planes used. I have used the plane function to create the cobble road and patio, along with the grass and stone brick

I have used the plane function to create the grass and stone planes along with the cobble road and patio which is lit by the streetlamps. The floor and ceiling of the café is also made with the plane function.



Procedural Box

For the procedural box I created a function that takes 3 parameters; width, height and depth which will then loop the cube faces code based on that data. I also created a scale variable of type double and set that to 1, this is used to calculate the positive vertex coordinates. With each loop the x, y or z variables increase each time so that when used to create the vertex coordinates the values change to create new cubes/faces with each repetition. I then render this the same as the previous functions. I use this function to create the benches and part of the tables in my scene as well as the café walls. This function along with the plane, and slab function show that I can make correctly lit and textured geometry which is also procedurally generated.

Streetlamp

To make the streetlamps in my scene I started off by making 4 procedurally generated boxes in my scene make function that is run once in the scene constructor at the start of the program to reduce lag when the program is running.

I then created a streetLampCreate function within the scene. In this I scale down the two boxes with the wood texture on the x and z axis so that they look more like a pole than a wooden box. I then used the translate function moving them to look like a streetlamp. I used the push and pop matrix function so that the translates used don't affect the other objects however I also use it to place 2 of the same objects in my scene reusing assets to save data and run speed. For the lamp I created 2 boxes one with the lamp off texture and the other with the lamp on texture. I translate these two in place using the same matrix. Within the scenes handle input I have it so if the h key is pressed it will make a bool variable true or false, this variable is used in an if statement to render the two boxes on and off depending on the value of the bool. Which makes it look like the streetlamps are turning on and off.

```
glPushMatrix();
glTranslatef(-0.25, 5, 1.25);//lamp
if (lampLights)
{
    streetLampOff.proceduralBoxRender();
}
else if(!lampLights)
{
    streetLampOn.proceduralBoxRender();
}
glPopMatrix();

glPopMatrix();
```

I have another function in scene which I created named streetLampRender in this I use the matrix stack to render four of the streetlamps using the streetlamp create function changing the location of the four streetlamps each time by translating and rotating it.

```
void Scene::streetLampRender()
{
    glPushMatrix();
    streetLampCreate();
    glPushMatrix();
    glTranslatef(14, 0, 0);
    streetLampCreate();
    glPopMatrix();
    glPushMatrix();
    glRotatef(180, 0, 1, 0);
    glTranslatef(-4, 0, -18);
    streetLampCreate();
    glPushMatrix();
    glTranslatef(-14, 0, 0);
    streetLampCreate();
    glPopMatrix();
    glPopMatrix();
    glPopMatrix();
}
```

Bench & Table

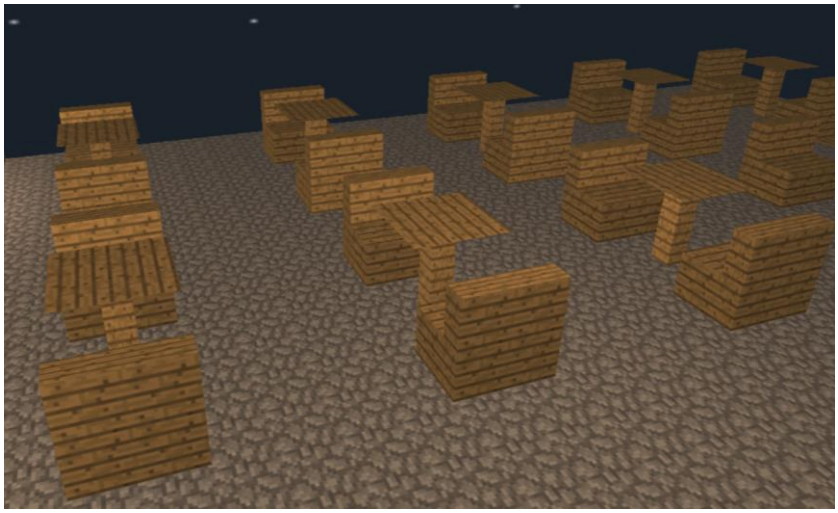
To create the table, I first made a procedural box and plane and set the texture to wood. I then scaled down the box by half in the x and z so that it looked like a table leg I then rendered the plane and translated it on top of the leg. I kept this creation within its own matrix using glPush and pop so that I can reuse the function in the scene and apply different transformations onto it.

```
void Scene::tableSetCreate()
{
    glPushMatrix();
    glPushMatrix();
    glTranslatef(45, 0, 1);
    glRotatef(270, 0, 1, 0);
    chairCreate();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(42, 0, 1.75);
    tableCreate();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(39.5, 0, 3);
    glRotatef(90, 0, 1, 0);
    chairCreate();
    glPopMatrix();
    glPopMatrix();
}
```

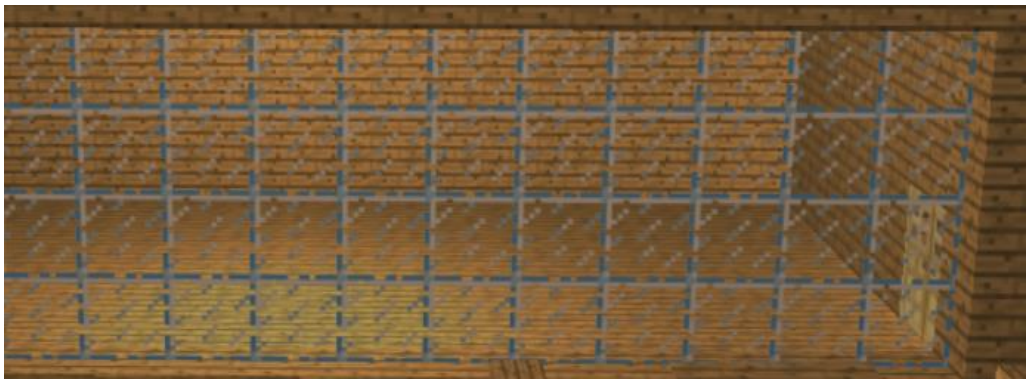
```
void Scene::tableSetRender()
{
    int z = 0;
    for (int i = 0; i < 5; i++)
    {
        z += 6;
        glPushMatrix();
        glTranslatef(0, 0, z);
        tableSetCreate();
        glPushMatrix();
        glTranslatef(-8, 0, 0);
        tableSetCreate();
        glPopMatrix();
        glPopMatrix();
    }
}
```



Window

To create the window, I first made a plane big enough to fit in the café wall I then set the window texture to it which I edited on photoshop to be transparent. To give the see-through effect of a window I use the open gl functions to blend the transparent parts of the plane into the background of the scene. To make this effective I must render the window last in the scene so that no other objects ruin the blending.

```
void Scene::windowRender()
{
    //window
    glPushMatrix();
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_BLEND);
    glRotatef(90, 1, 0, 0);
    glTranslatef(26, 5, -4.5);
    plane8.planeRender();
    glDisable(GL_BLEND);
    glPopMatrix();
}
```



Sky Box

As the skybox uses a lot of code and functions to work, I decided to make it its own class. To make the skybox work I use it in conjunction with the shape class to make the shape. For this I make a pointer of the Shape class and assign the skybox texture I made in photoshop to it. I then created a

vector that keeps track of the cameras position which is used to constantly translate the skybox to the cameras position. When rendering the skybox, I turn off open gl's depth testing which gives the effect of the skybox being far away. I also turn off the lighting so that the scenes lights don't affect the skybox making it brighter as that would seem unrealistic and ruin the emersion of the scene. Then in the scene I create a skybox pointer, initialise it and set the camera used in the skybox class to have the same data as the one used in the scene. Finally, I render it directly under the camera so that nothing can interfere with the depth testing.

```
void SkyBox::update()
{
    cubeCenter = camera->getPostion();//sets the cube center
    renderNew();
}

void SkyBox::renderNew()
{
    glPushMatrix();
    glTranslatef(cubeCenter.x, cubeCenter.y, cubeCenter.z);//
    glDisable(GL_DEPTH_TEST);
    glDisable(GL_LIGHTING);
    shape->renderCube();
    glEnable(GL_LIGHTING);
    glEnable(GL_DEPTH_TEST);
    glPopMatrix();
}
```

Lighting

For the lighting in my scene I have 8 lights as that is the maximum number allowed by opengl per scene. The first 4 lights are used for the streetlamps in my scene they are all spotlights of varying colours. To make them I create 3 glfloat variables for each light. One containing the lights colour value and brightness, the second holding the lights position and the third holding the directional value for the light. I then assign the variable data onto the light using opengl functions. I then assign the light a cut-off point which turns the light into a cone shape stopping light from rendering outside of the shape making it look like a streetlamp.

The next light I created is an ambient light for the scene to make it appear as the geometry is being slightly lit by the stars up above in the skybox. The light is having two glfloat variables one for the colour and brightness value and the other for the position. To create the light, I input the variable date into the light define and enable the light.

The next three lights are all roughly the same as they all have a similar purpose. They are used to light up the inside and outside of the café. One for the outside seating are one for the inside of the café and the last one is used to light up the bathroom. They each have two glfloat variables the first for light colour and brightness and the second for light direction. I then use open gl functions to assign that data onto the light. To make sure the light stays within the rooms I change the attenuation values of them.

Within the scene I have created a light class object so that I'll have access to the light class's functions. Within the scenes render function I render all the lights. However, I have set it up so that the lights are enabled and disabled when the H key is pressed. I did this in the scenes handle input

by using the input class to detect when the key is pressed. If it is it calls the enable lights function that I created in the lights class which is a simple switch statement. If the key is pressed again it calls the disable light functions. I have also made it so when the lights turn on and off the texture of the lamps change colour like in the game the lamps are from Minecraft. I do this by creating the lamps twice however assigning a different texture to each one then I use an if else statement when rendering the geometry that decides which lamp will be rendered. This is controlled by the lampLights Boolean also used to control the enabling and disabling of the lights so that the lights and textures are in sync.

```
//Lamp Lights
if (input->isKeyDown('h'))
{
    light.enableLight(0);
    light.enableLight(1);
    light.enableLight(2);
    light.enableLight(3);
    light.enableLight(5);
    light.enableLight(6);
    light.enableLight(7);

    lampLights = !lampLights;
    input->setKeyUp('h');
}
if (lampLights)
{
    light.disableLight(0);
    light.disableLight(1);
    light.disableLight(2);
    light.disableLight(3);
    light.disableLight(5);
    light.disableLight(6);
    light.disableLight(7);
}
```

Camera

To make the camera I created its own class with a constructor and destructor and many other functions. I created a bunch of vectors that will be used to calculate the cameras position. I also have three floats created to calculate the rotation of the camera.

```
Vector3 forward, up, right, position, lookAt, rotation;
float Pitch, Yaw, Roll;//used to calculate camera rotation
```

In the cameras constructor I simply set the input class pointer to equal the one in the parameters of the function. Which will be used in the scene to make sure the input data is transferred over to the camera class.

```
Camera(Input* in);
```

In the camera update function, I create some temporary variables and use them to calculate rotation with sin and cos. They are also used to calculate the vectors created in the header file.

```
float cosR, cosP, cosY;     //temp values for sin/cos from
float sinR, sinP, sinY;
//Roll, Pitch and Yall are variables stored by the camera
//handle rotation
cosP = cosf(Pitch * 3.1415 / 180);//x rotation
cosY = cosf(Yaw * 3.1415 / 180);//y rotation within 3d space
cosR = cosf(Roll * 3.1415 / 180);//z rotation
sinY = sinf(Yaw * 3.1415 / 180);
sinP = sinf(Pitch * 3.1415 / 180);
sinR = sinf(Roll * 3.1415 / 180);
```

I then create the array that will be used in the scene to set the cameras position. In the 0-2 spots I place the position vector, in the 3-5 spots I place the look at vector and in the 6-8 spots I place the up vector.

```
//used to update the camera in scene
gluLookAtArray[0] = { position.x };
gluLookAtArray[1] = { position.y };
gluLookAtArray[2] = { position.z };

gluLookAtArray[3] = { lookAt.x };
gluLookAtArray[4] = { lookAt.y };
gluLookAtArray[5] = { lookAt.z };

gluLookAtArray[6] = { up.x };
gluLookAtArray[7] = { up.y };
gluLookAtArray[8] = { up.z };
```

To control my camera, I created a keyboard input function that takes in the delta time from the scene through function parameter. In the function a create a temporary vector which will be used to calculate the different movements. If I want the camera to go forward I would press the W key the function would then recognise that input and assign the forward vector to the temporary vector scale it down to the appropriate speed and update the position vector to add the temporary vector to it so that the camera moves forward. I then use roughly the same code for the other movements but replace the vector with the appropriate one for example if I wanted to go right id use the right vector to calculate it.

```
int speed = 1;
double cameraSpeed = 0.6;
Vector3 tempVec;
if (input->isKeyDown('w'))//forward
{
    tempVec = forward;
    tempVec.scale(cameraSpeed);
    position += tempVec;
    input->setKeyUp('w');
}
```

To have mouse control the cameras rotation I created a function that takes delta time, and the screens width and height as parameters. The first thing I do in the function is reset the mouse so I can calculate the rotations properly. Using the mouse's position on the screen I work out the distance between that and the central point and use it to update the yaw and pitch vectors, this makes it so when the mouse is moved up the camera will rotate up. I then set the mouse's position to be the centre of the screen I calculated this using the parameters inputted into the function. This resets the mouse so that the next rotation can be calculated. I get the data used to calculate the mouse movement from code used in the scene. Within the scenes resize function I set the mouse's position on the screen using the width and height variables in the function.

```
void Camera::mouseInput(float dt, float width, float height)
{
    glutSetCursor(GLUT_CURSOR_NONE);//resets mouse

    screenWidth = width;
    screenHeight = height;

    float mouseDifferenceX, mouseDifferenceY, centrePositionX, centrePositionY;
    float scale = 55.0;//used to scale down the mouse movement speed

    centrePositionX = screenWidth / 2;
    centrePositionY = screenHeight / 2;

    mouseDifferenceX = input->getMouseX() - centrePositionX;//calculates how much the mouse should be moved left or right
    mouseDifferenceY = input->getMouseY() - centrePositionY;//calculates how much the mouse should be moved up or down

    Yaw = Yaw + (mouseDifferenceX * scale * dt);//up and down
    Pitch = Pitch - (mouseDifferenceY * scale * dt);//left and right
    glutWarpPointer(centrePositionX, centrePositionY);//sets mouse position to be the middle of the screen

    //mouse code in scene cpp resize and constructor
}
```

Conclusion

Overall, I believe I have made a well lit and textured 3d scene made from geometry=try created by myself. The camera I created functions well, and the scene looks like it could belong in the game Minecraft. I would have liked to have been able to add a mirror in the bathroom however I couldn't get it working in sync with the blending on the doors and window. If I had more time, I would have also added a second camera and shadows.

References

As all the textures used are from the game Minecraft, I have used the software as my only reference.

Persson, M., 2011. *Minecraft*. Sweden: Mojang.