CSU44052 Final Project Report

Name:	Róisín Ní Bhriain
Student ID:	18326577
Declaration:	The work on both the programming and written assignments are entirely my own and I have not collaborated with anyone.
Youtube link 1: Required Features	https://youtu.be/glljHoDj1b0
Youtube link 2: Final Demo	https://youtu.be/pdfePmAXa3g

Required feature 1: crowd of animated snow-people/reindeer/robins etc.

Screenshot(s) of feature:

Describe how you implemented it: Each snowman is a hierarchical model with both a spinning hat and waving arms. They currently translate forwards and backwards in a line. There are 9 of these snowmen.

Pseudocode:

```
for (int i = 0; i < 9; i++) {
  translate (snowman, vec3(-75.0f+i*20, 0.0f, forward_z));
  drawSnowmanBody();
  arms = rotate_z_deg(arms, rotate_z);
  arms = snowman* arms;
  drawArms();
  rotate_y_deg(hat, angle);
  hat = snowman * hat;
  drawHat();
}
```

Credits (e.g., list source of any tools, libraries, assets used):

- https://www.turbosquid.com/3d-models/sir-snowman-toy-model-1343310

Required feature 2: texture-mapping your scene and creatures using an image file

Screenshot(s) of feature:

- Hat mapped with black texture
- Snowman body mapped with snow texture

- Arms mapped with wood texture
- All other objects are mapped with different textures

Describe how you implemented it: I used the stb image library to load each texture into an unsigned int in the init function. Then for each object using a texture I made sure to bind the required texture before drawing the object.

Pseudocode:

```
LoadTexture();
draw () {
  glEnable(GL_TEXTURE_2D);
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, snowTexture);
  drawArrays();
}
```

Credits (e.g., list source of any tools, libraries, assets used):

- Stb_image.h
- Google for jpg pictures such as: wood, bark, snow textures
- https://learnopengl.com/Getting-started/Textures

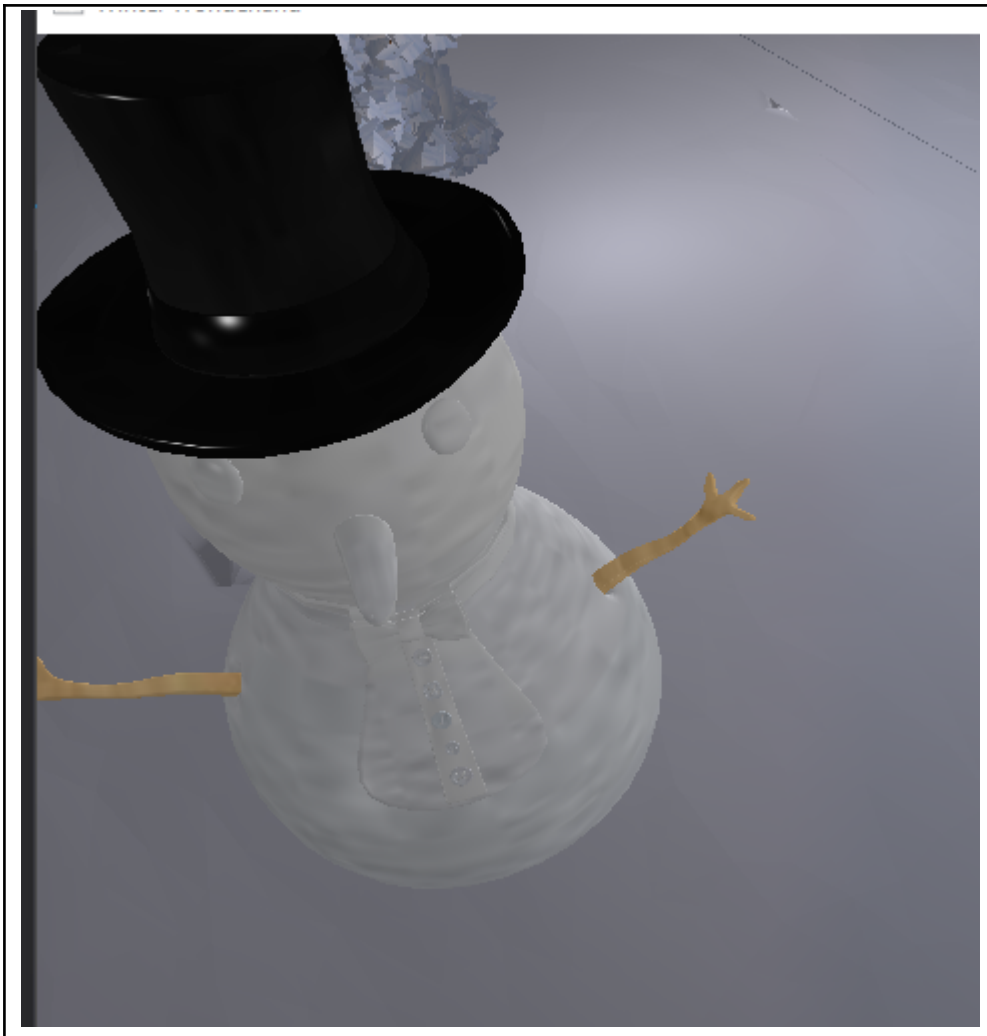
Required feature 3: implementation of the Phong Illumination model

- **a.** Multiple light sources (at least 2, can be point, directional, or spotlight)
- **b.** Multiple different material properties (at least 5 on 5 different objects)
- **c.** Normal must be transformed correctly
- **d.** Shading must use a combination of ambient, diffuse, and specular lighting

Screenshot(s) of feature:

Point lights

Different materials



Describe how you implemented it: This was mostly done in the shader. The locations of the light sources are set up in the main. The normal was transformed using the eye position and the fragment position in the vertex shader. There are several point lights from different positions and one directional light.
Each material is set with different ambient, diffuse and specular values. This is demonstrated in the video. The calculated light sources are all added to the result.

Pseudocode:

- in the vertex shader

```
vec3 tnorm = normalize( NormalMatrix * vertex_normal);
```

- two different lights

```
vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir);
vec3 CalcPointLight(PointLight light, vec3 normal, vec3 fragPos, vec3 viewDir);
```

- for each object

```
setAmbience()
setDIffuse()
setSpecular()
setShininess()
```

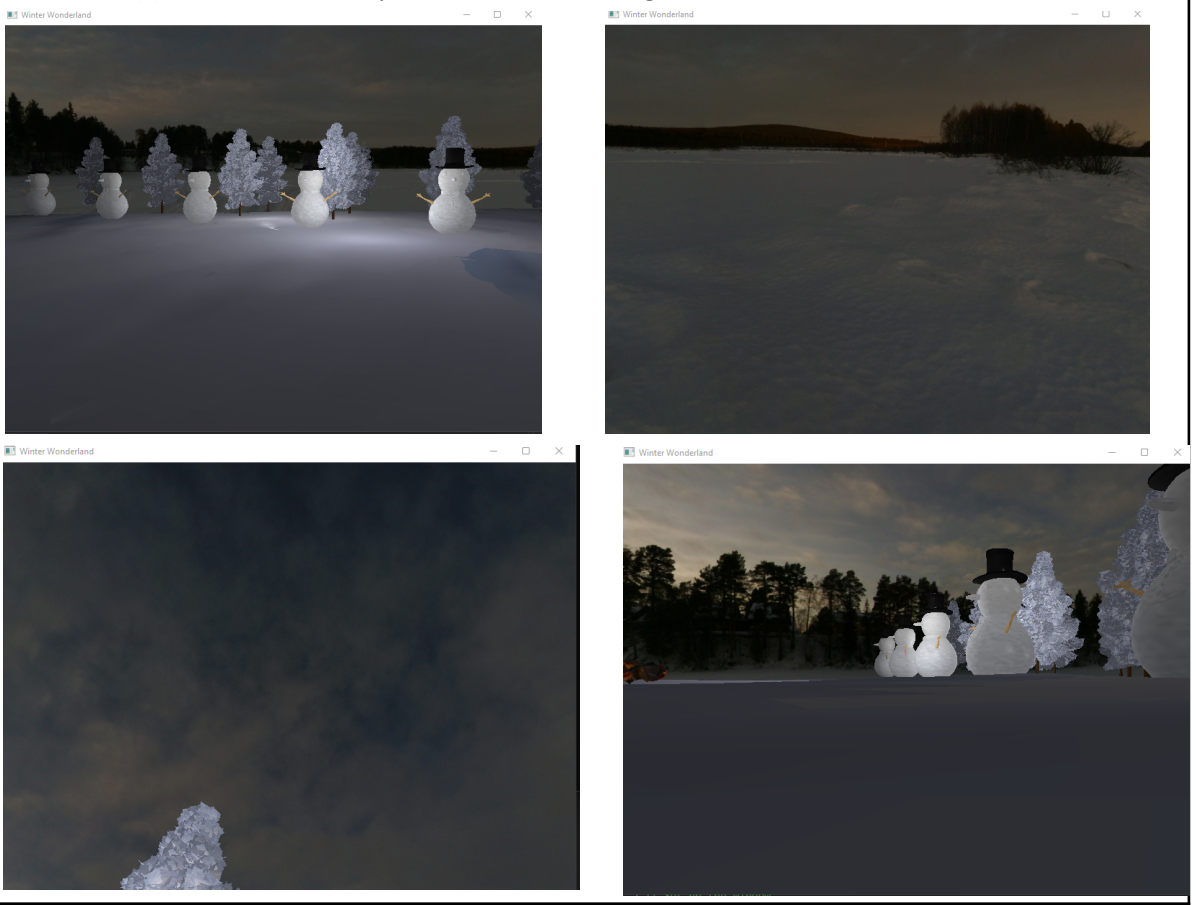
Credits (e.g., list source of any tools, libraries, assets used):

- https://learnopengl.com/Lighting/Basic-Lighting

Advanced Feature 1:

Description/name of feature: Cube-mapping for a Skybox

Screenshot(s) of feature: The skybox from different angles.



Describe how you implemented it: First I created a separate shader for the skybox along with a separate class. I loaded in the texture by using a loadCubemap function and stored these faces as a texture in the skybox class. Using the vertices for the skybox I generated the object buffer and bound the vao and vbo. Finally, the draw function binds the vao and the cube mapped texture and draws the vao (36 triangles). In the main the skybox uses its own camera. This way the camera rotations are applied in the same way as the viewer camera but the camera transformations are not applied. I also make sure to set the Depth function as GL_LEQUAL so the skybox is infinite.

Pseudocode:

```
genObjectBuffer();
loadCubemapTexture();
draw () {
  skyboxShader.use();
  glDepthFunc(GL_LEQUAL);
  drawSkybox();
}
```

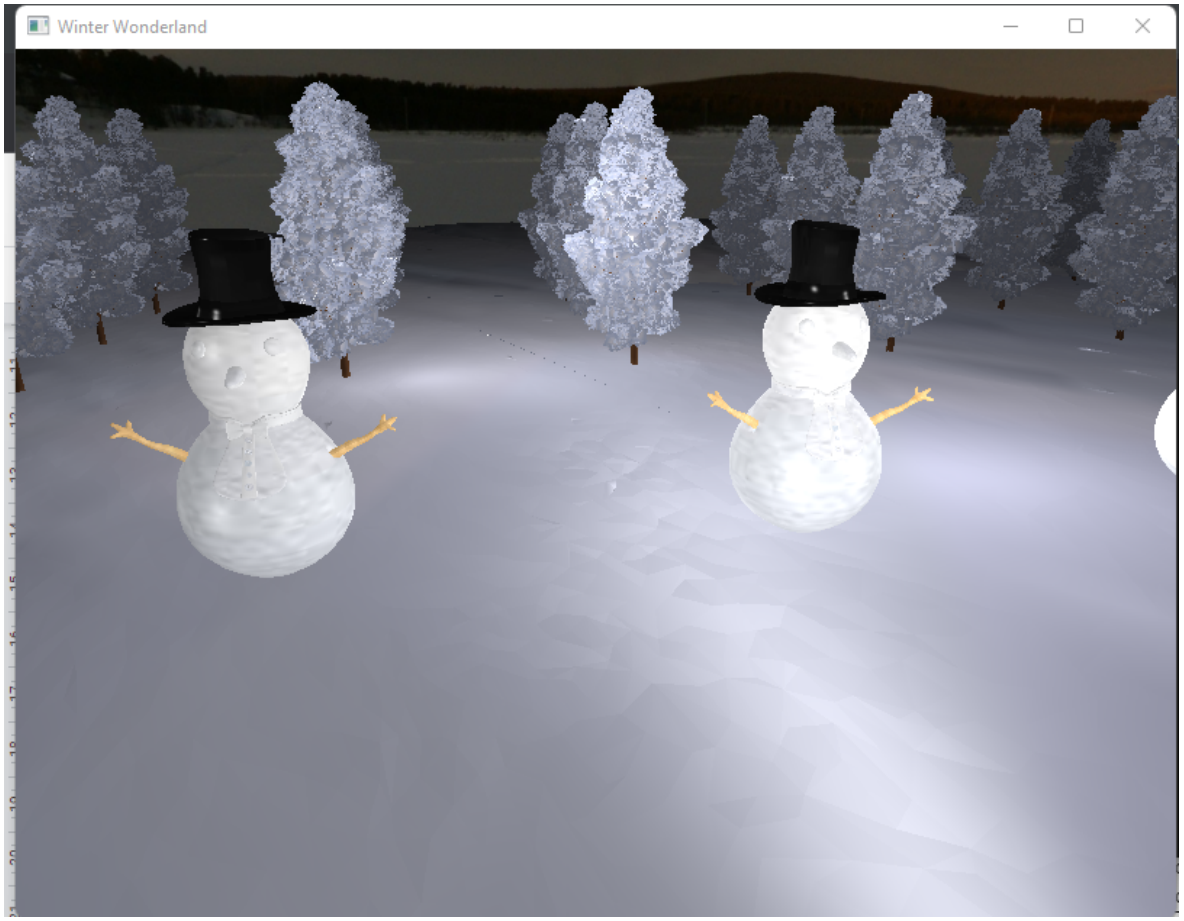
Credits (e.g., list source of any tools, libraries, assets used):

- Skybox images - https://opengameart.org/content/winter-skyboxes?page=1
- Stb_image.h
- https://learnopengl.com/Advanced-OpenGL/Cubemaps

Advanced Feature 2:

Description/name of feature: Hierarchical Crowd - walking snowmen

Screenshot(s) of feature:



Describe how you implemented it: This was done using several rotates and a translate. The snowmen translate back and forth. They also spin on the z axis and the y axis to make it look like they're working.

Pseudocode:

- in display

```
mat4 snowman = identity_mat4();
                snowman = rotate_y_deg(snowman, rotatingSnowMan);
                snowman = rotate_y_deg(snowman, spin);
                snowman = rotate_z_deg(snowman, steps);
                if (on) {
                        snowman = rotate_y_deg(snowman, 180);
                        snowman = translate(snowman, vec3(-75.0f + (i - 1) * 10, 0.0f,
-forward_x));
                }
                else {

                        snowman = translate(snowman, vec3(-75.0f + i * 10, 0.0f, 20+ forward_x));
```

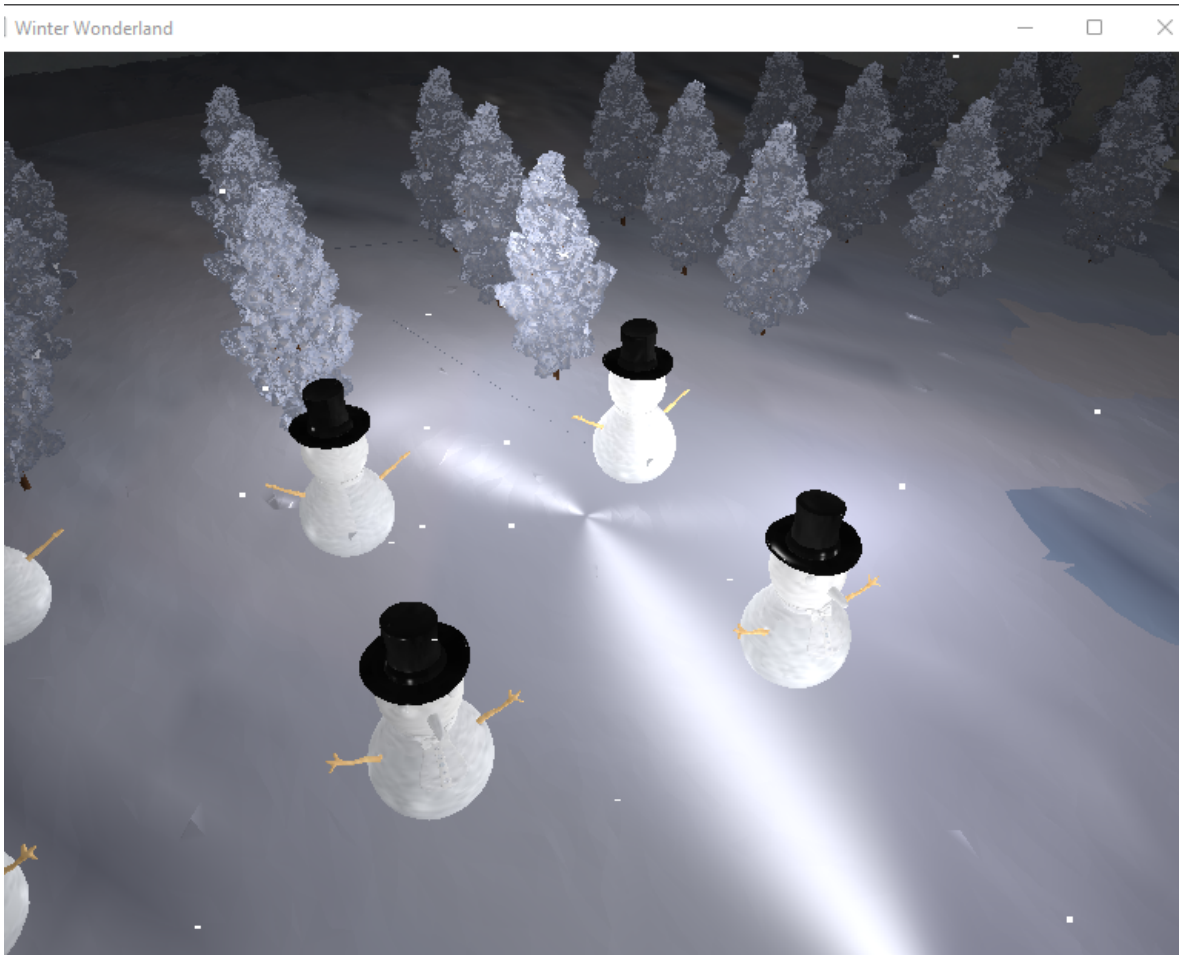
```
                    }
    -   in update scene
if (forward_x > 20.0f || forward_x < -20.f) {
                        if (turned) {
                                y = -y;
                                //direction = -direction;
                                turned = !turned;
                        }
                        else {
                                spin+= 1.0f;
                                if (fmodf(spin,   90.0f) == 0) {

                                        turned = !turned;
                                }
                        }
                }
                forward_x += y * delta;
                angle += 1.0f;
                steps += direction;

                // snowman stepping
                if (rotatingSnowMan > 10.0f || rotatingSnowMan < -10.0f) {
                        direction = -direction;
                }
                rotatingSnowMan += direction;
```

Credits (e.g., list source of any tools, libraries, assets used):

Advanced Feature 3:

Description/name of feature: Snow animation

Screenshot(s) of feature:



This shows up better in the video but there are some flakes to be seen here.

Describe how you implemented it: This is implemented by using GL Lines. An array of “snowflakes” are created using random locations for the initial x and y. There is also a random radius and a random increment speed for each snowflake.

Pseudocode:

```
for (int i = 0; i < snowSize(); i++) {
  initSnowFlake();
}
drawSnow () {
  for (int i = 0; i < snowSize(); i++) {
    drawSnowFlake();
  }
}
display () {
  drawSnow();
```

}

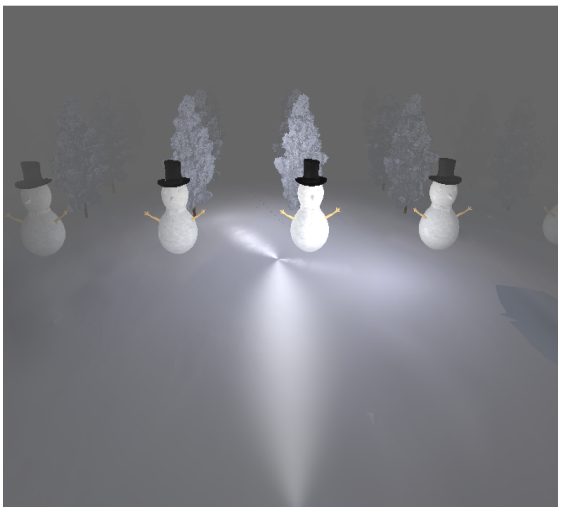
Credits (e.g., list source of any tools, libraries, assets used):

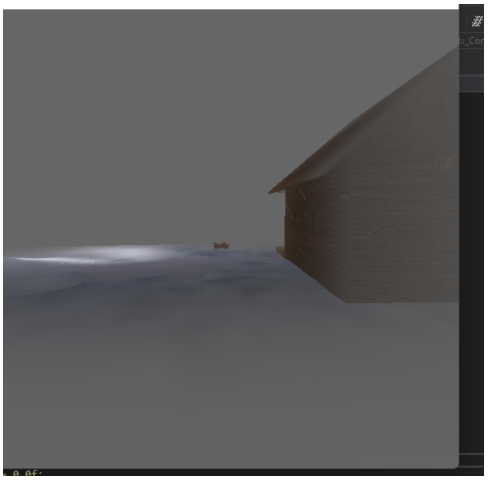
- https://stackoverflow.com/questions/53814299/creating-rain-drops

Advanced Feature 4:

Description/name of feature: Fog

Screenshot(s) of feature:





Describe how you implemented it: This was done in the fragment shader. It uses an exponential function which uses fog density, fragment distance from camera and the gradient. Both the density and the fog were manually set to a value that looked good. This colour was then mixed in with the fragment colour. It requires that the fog be the same colour as the sky so it blends correctly.

Pseudocode:

```
float getFogFactor()
{
    float dist = distance(position_eye, FragPos);
        float result = 0.0;
        result = exp(-pow(density * dist, 3.0));

        result = clamp(result, 0.0, 1.0);
        return result;
}
gl_FragColor = mix(fog_colour, gl_FragColor, getFogFactor());
```

Credits (e.g., list source of any tools, libraries, assets used):

- https://opengl-notes.readthedocs.io/en/latest/topics/texturing/aliasing.html

Advanced Feature 5:

Description/name of feature: Snowglobe (alpha blending)

Screenshot(s) of feature:

Describe how you implemented it: I did this using a dome shaped mesh which I placed on the two snowmen in the picture. The actual alpha blending was done in the fragment shader. I added a new uniform to the fragment shader as a transparency variable. For the normal objects this remained as a 1 and for the dome I made this a 0.25. This meant I had to make sure that the dome was drawn last so that the objects could be seen through the dome. The y value is decremented each time it is drawn so that it continues “falling”. When the y value reaches a certain number it is restarted from the top of the screen.

Pseudocode:

```
planeShader.setFloat("transparency", 0.25);
drawDome();
```

- in fragment shader

```
uniform float transparency;
gl_FragColor = vec4(result, transparency);
```

Credits (e.g., list source of any tools, libraries, assets used):