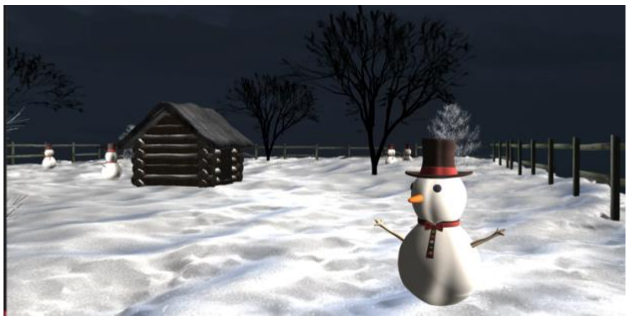
CSU44052 Final Project Report

Name:	Kinjal Bhattacharyya
Student ID:	21344426
Declaration:	**I understand that this is an individual assessment and that collaboration is not permitted. I have not received any assistance with my work for this assessment on both programming and written assignments. Where I have used the published work of others, I have indicated this with appropriate references.** **I have not and will not share any part of my work on this assessment, directly or indirectly, with any other student.** **I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.** **I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write."** **I understand that by returning this declaration with my work, I am agreeing with the above statement.** **Name: Kinjal Bhattacharyya** **Date: 06-01-2023**
Youtube link 1: Required Features	https://youtu.be/FWDOMHYW4_A
Youtube link 2: Final Demo	https://youtu.be/oIJOUaVp9IA
GitHub Link: **Code Repo**	https://github.com/Kinjal-16/CGraphics.git

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

Screenshot(s) of feature:



A crowd of snowperson dispersed throughout the scene

Describe how you implemented it:

1. A crowd of snowmen has been animated in this animation.
2. Each snowman, although identical, were processed individually and therefore had their own separate VAOs, VBOs, and separate instances of the shader class declared for each of them.
4. The starting position in the scene as well the orientation(along the y-axis) is given separately for each snowmen. That's why the snowmen are dispersed throughout the scene.
4. A loop is rum from 0 to 5(for 6 snowmen) and the snowmen are drawn.
5. The description of how the snowmen are translating and rotating in the scene are given in Advanced feature under Advanced animation

Pseudocode:

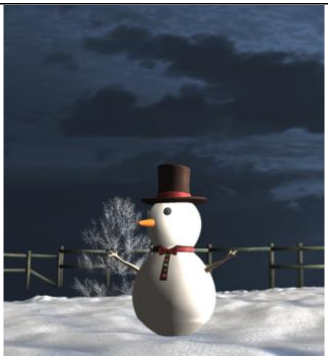
```
 Step 1: - Store the names of the models in an array
Step 2 : - Declare instances of the shader class equal to the size of the crowd
Step 3: - Initialise the translation and rotation vectors for each model with unique values to make them look dispersed throughout the scene
Step4 :  - Loop from 0 to n-1, where n is the size the crowd
Step 5:- In the loop, before calling the draw function for each model, check for collisions and animate the snowman by translating and rotating them with a predefined algorithm(Discussed in advanced feature)
Step 6: - Draw each model with the given model position
```

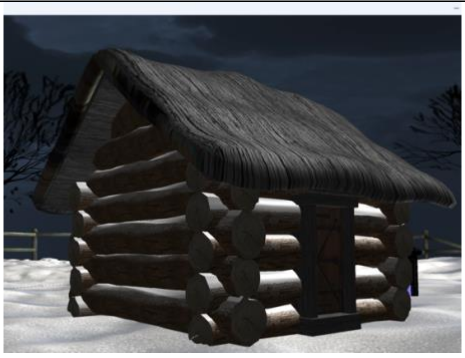
Credits (e.g., list source of any tools, libraries, assets used):

Models:-
Snowman :- 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:

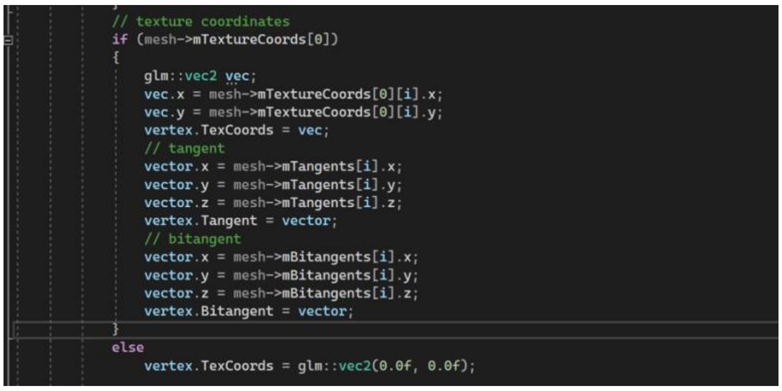

Texture-mapping of snowman


Texture mapping of the hut and the scene

Describe how you implemented it:

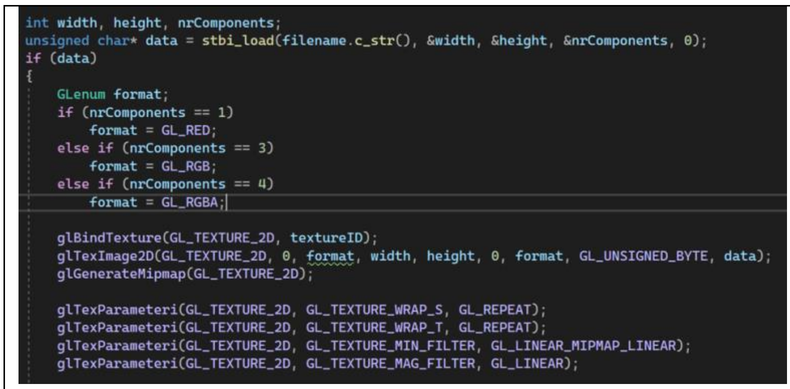
1. Import the model using the assimp library
2. The model has texture co-ordinates given in them
3. Extract these texture coordinates and export them to the fragment shader which deals with colours and textures via the vertex shader.
3. A uniform of type sampler2D is declared in the fragment shader which tells the fragment shader the texture(/s) to use on the object.
4. The stb_library is used to read the texture image itself form the given location. If the model is of type obj, the corresponding .mtl file has the location of the texture image
5. This texture object is then bound and then can be accessed by the uniform in fragment shader of type sampler2D.
6. The fragment shader now has both the texture coordinates and the corresponding textures object. The output is therefore the textures drawn on the object using the fragment shader.
7. A minmap is also generated to deal with changing resolution of an object in the scene

Pseudocode:



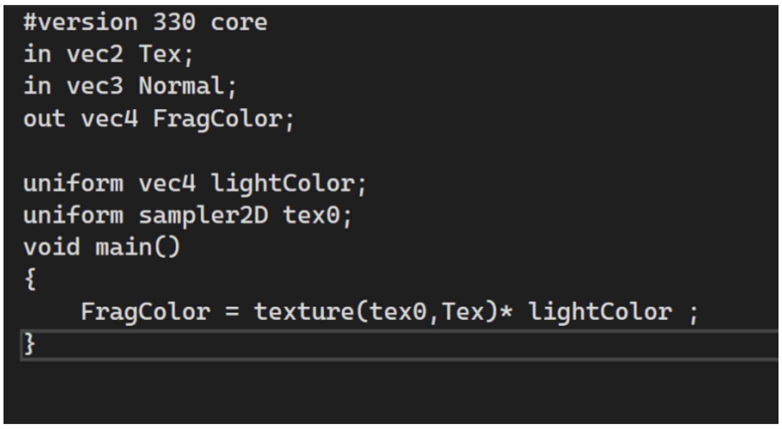
```
// texture coordinates
if (mesh->mTextureCoords[0])
{
    glm::vec2 vec;
    vec.x = mesh->mTextureCoords[0][i].x;
    vec.y = mesh->mTextureCoords[0][i].y;
    vertex.TexCoords = vec;
    // tangent
    vector.x = mesh->mTangents[i].x;
    vector.y = mesh->mTangents[i].y;
    vector.z = mesh->mTangents[i].z;
    vertex.Tangent = vector;
    // bitangent
    vector.x = mesh->mBitangents[i].x;
    vector.y = mesh->mBitangents[i].y;
    vector.z = mesh->mBitangents[i].z;
    vertex.Bitangent = vector;
}
else
    vertex.TexCoords = glm::vec2(0.0f, 0.0f);
```

a. Code for extracting the texture coordinated from the model



```
int width, height, nrComponents;
unsigned char* data = stbi_load(filename.c_str(), &width, &height, &nrComponents, 0);
if (data)
{
    GLenum format;
    if (nrComponents == 1)
        format = GL_RED;
    else if (nrComponents == 3)
        format = GL_RGB;
    else if (nrComponents == 4)
        format = GL_RGBA;

    glBindTexture(GL_TEXTURE_2D, textureID);
    glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format, GL_UNSIGNED_BYTE, data);
    glGenerateMipmap(GL_TEXTURE_2D);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

b. Code for loading the texture object and binding it



```
#version 330 core
in vec2 Tex;
in vec3 Normal;
out vec4 FragColor;

uniform vec4 lightColor;
uniform sampler2D tex0;
void main()
{
    FragColor = texture(tex0,Tex)* lightColor ;
}
```

c. Code for fragment shader that draws the texture on the object

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

1. References: -
https://learnopengl.com/Model-Loading/Model
2. Models: -
Snowman :- https://www.turbosquid.com/3d-models/sir-snowman-toy-model-1343310

Fence :- https://www.turbosquid.com/3d-models/3d-model-old-wooden-fence-gate-1696345

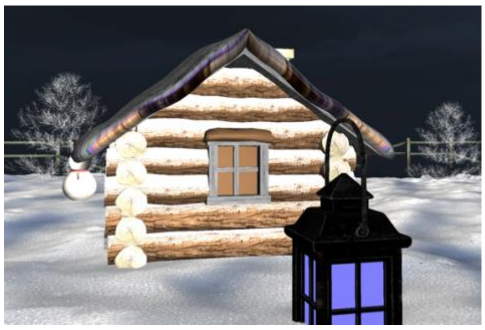
Hut: - https://sketchfab.com/3d-models/snowy-wooden-hut-857c02ab61834df8a8d125ccce6fec2a

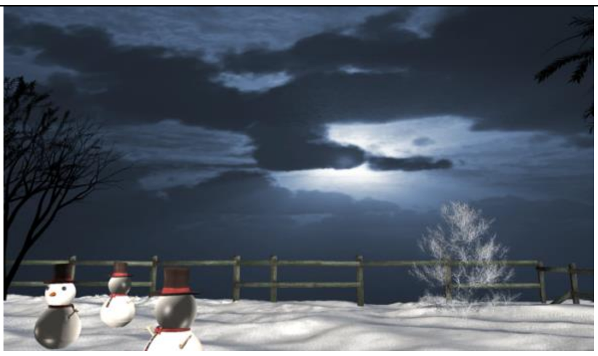
Base: - https://sketchfab.com/3d-models/snow-raw-scan-67cdd845964e4614a818f35f55de9622

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:


a. Point light


b. Directional light

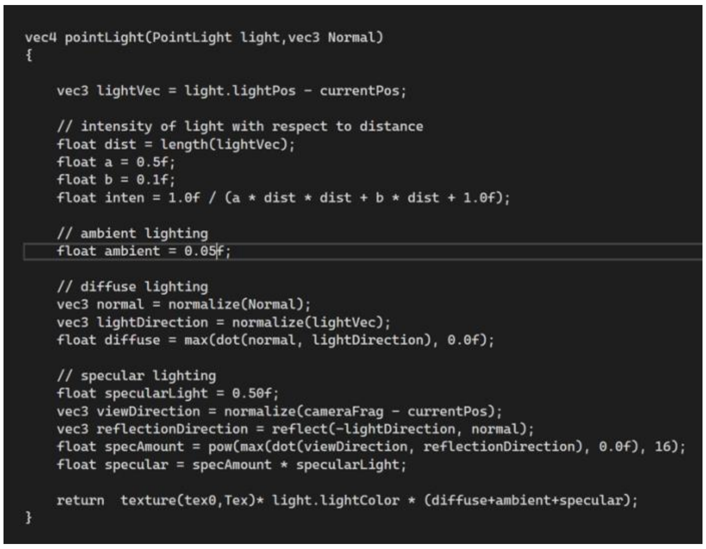
Describe how you implemented it:

a. I have implemented directional light to show the effect of the moon in the scene and point lighting for the lantern in the scene
b. Material properties for multiple models is extracted using assimp including:

1. The Snow Base
2. The fence around the base
3. The snowman(Each of them)
4. The Hut
5. The trees

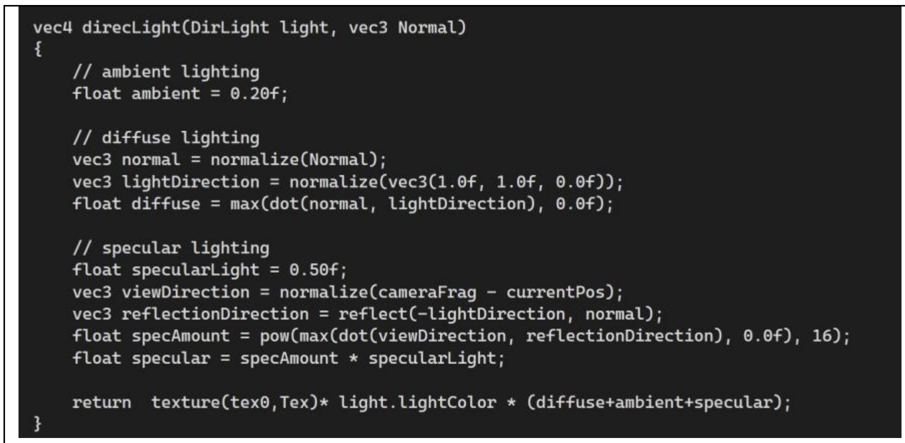
c. Normals are extracted from each model and exported into the fragment shader to be transformed correctly.
d. Ambient, Diffuse and specular lighting is implemented for both the light sources and the added effect of all three is taken as output

Pseudocode:



```
vec4 pointLight(PointLight light,vec3 Normal)
{

    vec3 lightVec = light.lightPos - currentPos;

    // intensity of light with respect to distance
    float dist = length(lightVec);
    float a = 0.5f;
    float b = 0.1f;
    float inten = 1.0f / (a * dist * dist + b * dist + 1.0f);

    // ambient lighting
    float ambient = 0.05f;

    // diffuse lighting
    vec3 normal = normalize(Normal);
    vec3 lightDirection = normalize(lightVec);
    float diffuse = max(dot(normal, lightDirection), 0.0f);

    // specular lighting
    float specularLight = 0.50f;
    vec3 viewDirection = normalize(cameraFrag - currentPos);
    vec3 reflectionDirection = reflect(-lightDirection, normal);
    float specAmount = pow(max(dot(viewDirection, reflectionDirection), 0.0f), 16);
    float specular = specAmount * specularLight;

    return  texture(tex0,Tex)* light.lightColor * (diffuse+ambient+specular);
}
```

a. Point light code in the fragment shader showing normals transformed correctly and shading with ambient, diffuse and specular



```
vec4 direcLight(DirLight light, vec3 Normal)
{
    // ambient lighting
    float ambient = 0.20f;

    // diffuse lighting
    vec3 normal = normalize(Normal);
    vec3 lightDirection = normalize(vec3(1.0f, 1.0f, 0.0f));
    float diffuse = max(dot(normal, lightDirection), 0.0f);

    // specular lighting
    float specularLight = 0.50f;
    vec3 viewDirection = normalize(cameraFrag - currentPos);
    vec3 reflectionDirection = reflect(-lightDirection, normal);
    float specAmount = pow(max(dot(viewDirection, reflectionDirection), 0.0f), 16);
    float specular = specAmount * specularLight;

    return  texture(tex0,Tex)* light.lightColor * (diffuse+ambient+specular);
}
```

b. Point light code in the fragment shader showing normals transformed correctly and shading with ambient, diffuse and specular

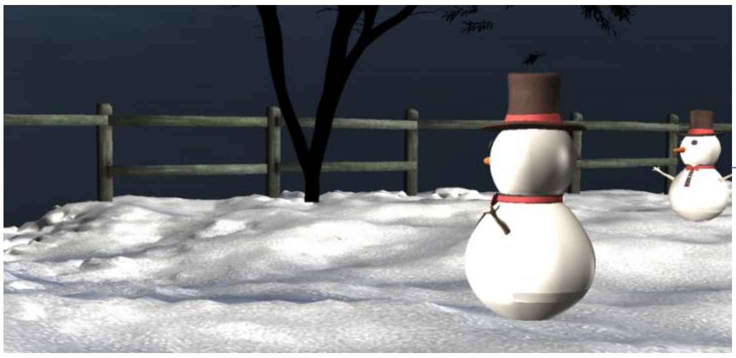
Credits (e.g., list source of any tools, libraries, assets used):

References: -
https://github.com/VictorGordan/opengl-tutorials.git

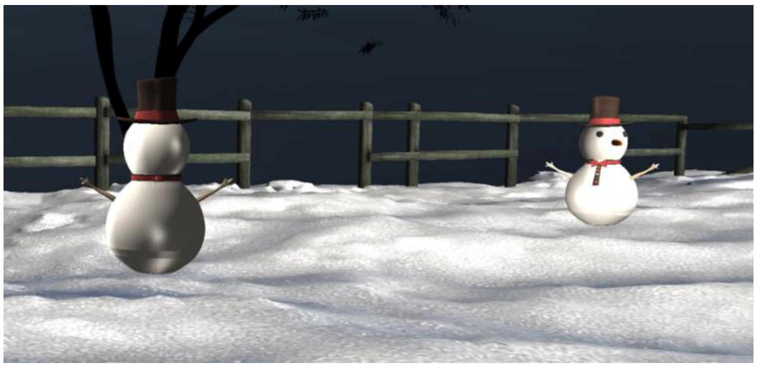
Advanced feature 1

Description/name of feature: Collision detection among snowmen as well as a snowman and the fence.

Screenshot(s) of feature:



a. The snowman in the back is about to collide with fence

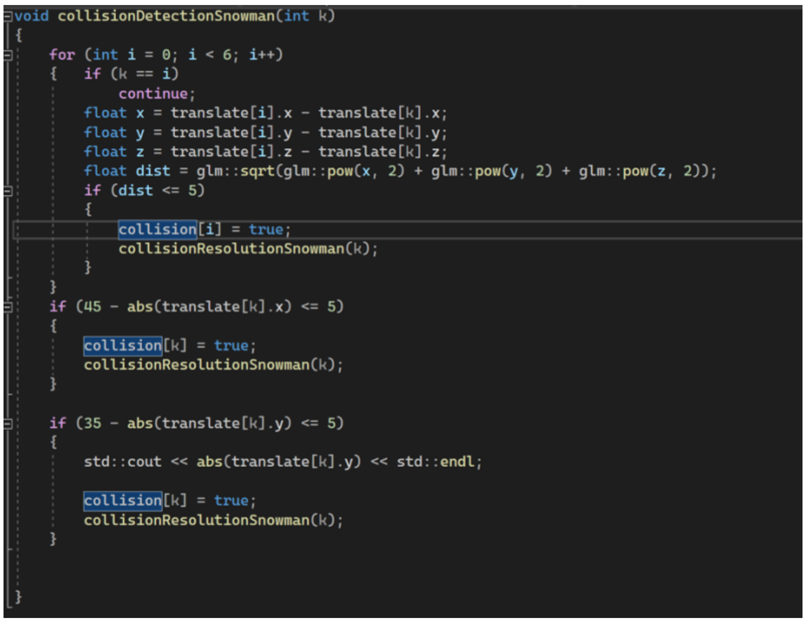


b. It rotates here to avoid the collision

Describe how you implemented it:

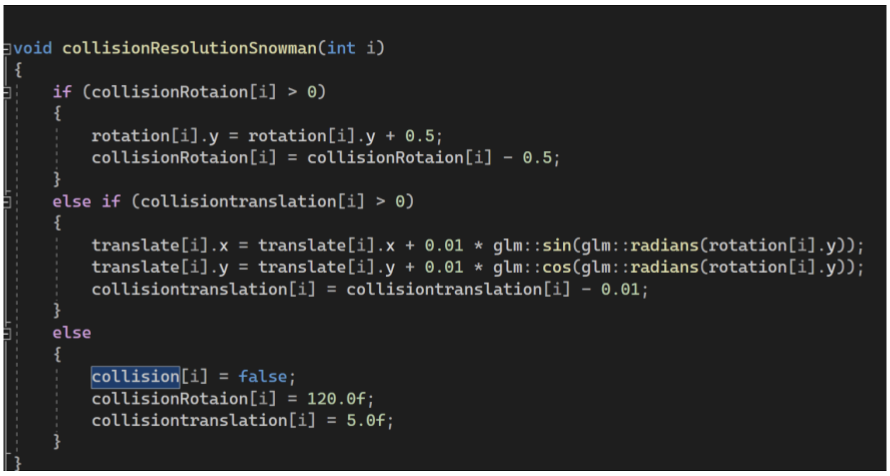
a. Every time a snowman is getting drawn, invoke the collision detection function to check for collision or the collision resolution function if a collision is already detected.
b. Inside this function, the distance between the centre of the snowman from the centre of every other snowman in the crowd is calculated to detect snowman to snowman collision.
The distance between the centre of the snowman and the 4 planes that forms the fence around the boundary is also calculated.
c. If this distance is smaller than 5 units, a flag is raised to indicate to the main function that a collision has been detected and the collision resolution function is invoked
d. The collision resolution function gradually rotates the snowman by 120 degrees and translates it 5 units along the direction the snowman is facing. The way this translation happens is given under Advanced feature 2. Once this is done, the collision detected flag is negated and the snowman moves normally in the scene.

Pseudocode:



```
void collisionDetectionSnowman(int k)
{
    for (int i = 0; i < 6; i++)
    {   if (k == i)
            continue;
        float x = translate[i].x - translate[k].x;
        float y = translate[i].y - translate[k].y;
        float z = translate[i].z - translate[k].z;
        float dist = glm::sqrt(glm::pow(x, 2) + glm::pow(y, 2) + glm::pow(z, 2));
        if (dist <= 5)
        {
            collision[i] = true;
            collisionResolutionSnowman(k);
        }
    }
    if (45 - abs(translate[k].x) <= 5)
    {
        collision[k] = true;
        collisionResolutionSnowman(k);
    }

    if (35 - abs(translate[k].y) <= 5)
    {

        std::cout << abs(translate[k].y) << std::endl;

        collision[k] = true;
        collisionResolutionSnowman(k);
    }


}
```

a. The collision detection function



```
void collisionResolutionSnowman(int i)
{
    if (collisionRotaion[i] > 0)
    {
        rotation[i].y = rotation[i].y + 0.5;
        collisionRotaion[i] = collisionRotaion[i] - 0.5;
    }
    else if (collisiontranslation[i] > 0)
    {
        translate[i].x = translate[i].x + 0.01 * glm::sin(glm::radians(rotation[i].y));
        translate[i].y = translate[i].y + 0.01 * glm::cos(glm::radians(rotation[i].y));
        collisiontranslation[i] = collisiontranslation[i] - 0.01;
    }
    else
    {
        collision[i] = false;
        collisionRotaion[i] = 120.0f;
        collisiontranslation[i] = 5.0f;
    }
}
```

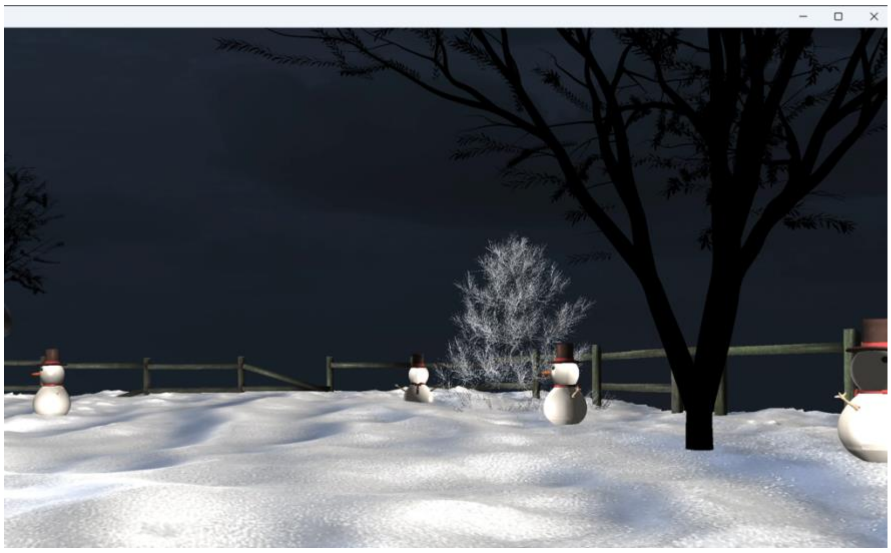
b. The Collision Resolution function

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

Advanced feature 2

Description/name of feature: Animation

Screenshot(s) of feature:



Animated Snowmen

Describe how you implemented it:

1. Take the angle the snowman makes with respect to the y axis
2. To make the snowman move along the direction that it faces , the following formulae is applied:-

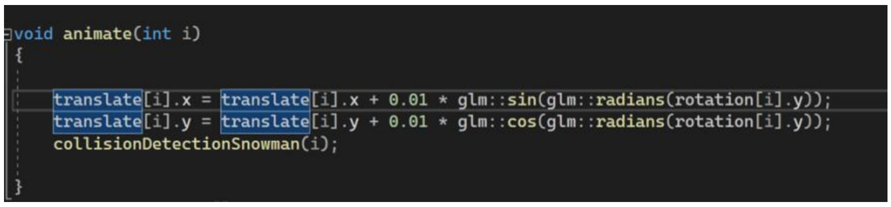
X = X + Cos theta * 0.01

Y = Y + Sin theta * 0.01

Here 0.01 is the constant value by which we are translating the snowman along X and Y axis. We are effectively moving the snowman by 0.01 units with every iteration but in the X-Y plane making it more realistic than animation along a single axis.

We are also calling the collision detection function via the animation function to ascertain that the translation wouldn't cause the model to collide with another.

Pseudocode:



```
void animate(int i)
{

    translate[i].x = translate[i].x + 0.01 * glm::sin(glm::radians(rotation[i].y));
    translate[i].y = translate[i].y + 0.01 * glm::cos(glm::radians(rotation[i].y));
    collisionDetectionSnowman(i);

}
```

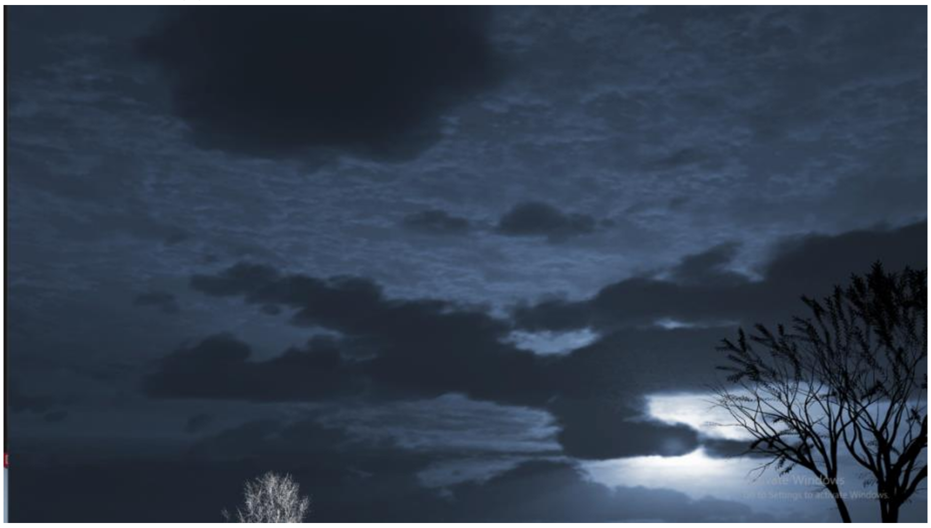
Animation

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

Advanced feature 3

Description/name of feature: Cube Mapping

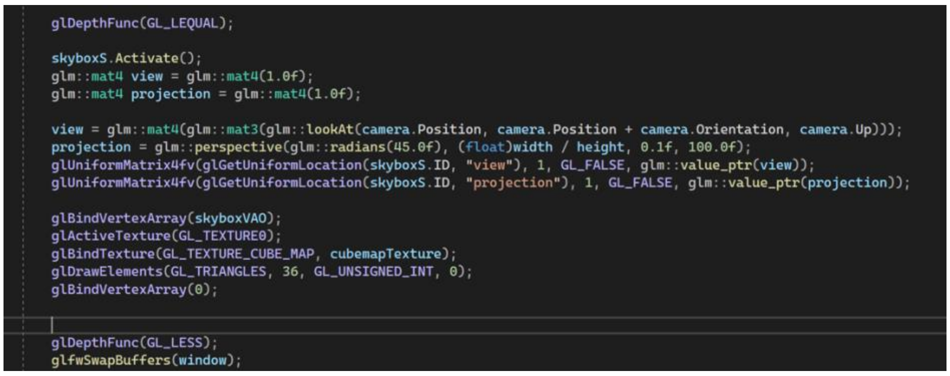
Screenshot(s) of feature:



Describe how you implemented it:

1. Took an HDRI image of a sky and converted it into cube map
2. Read the 6 images generated, each for a face of the cube from the cube map.
2. Declared the vertices and indices for it
3. Read each texture and putting them in the cube texture
4. Disabled vertical flipping
5. Wrote the shaders for the cube map and export the texture units
6. Draw it on the screen

Pseudocode:



```
glDepthFunc(GL_LEQUAL);

skyboxS.Activate();
glm::mat4 view = glm::mat4(1.0f);
glm::mat4 projection = glm::mat4(1.0f);

view = glm::mat4(glm::mat3(glm::lookAt(camera.Position, camera.Position + camera.Orientation, camera.Up)));
projection = glm::perspective(glm::radians(45.0f), (float)width / height, 0.1f, 100.0f);
glUniformMatrix4fv(glGetUniformLocation(skyboxS.ID, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(skyboxS.ID, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

glBindVertexArray(skyboxVAO);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0);
glBindVertexArray(0);


glDepthFunc(GL_LESS);
glfwSwapBuffers(window);
```

Drawing the skybox

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

https://assetstore.unity.com/packages/2d/textures-materials/sky/skybox-series-free-103633
https://learnopengl.com/Advanced-OpenGL/Cubemaps