# Museum at night

Amir Akintunde Sanni

CMP203

## Overview:

For this Project I created a museum like Scene set at night with assets gotten from popular forms of media including Games, Movies, Shows and also assets provided from the starting project, most external assets gotten from an archive of game assets from different generations of gaming hardware (Link). All Techniques used in this project were learned and applied with the help of the video lectures and slides provided for this course.

This project was made in C++ while using the Legacy OpenGL graphics library and the irrklang Audio Library which allowed me to implement 3d audio.

## Controls:

### Main Camera

Controlled using W, A, S, D for Movement and Z to move upward on the Y axis and X to move downward on the Y axis.

The user is also able to control where the camera looks by using their mouse note that the controls to look on the x axis are inverted ("annoying I know but that's what I'm used to").

W: Move Forward

A: Move Backward

S: Move Right

D: Move Left

Mouse movement to Look in desired location.

### Static Camera

The user can control what camera they want to look at by pressing.
numpad 1:  to switch to the first static camera.

Numpad 2: to switch to the Second static camera.

Numpad 3: to switch to the Third static camera.

F: To switch to the main movable camera.

## Rotation of Reflecting Object

The user has the ability to control the rotation on the Y axis of the asset.

q: Change rotation of Reflecting Object

### general user interaction

r: To switch to the wire frame view (NOTE: to switch back to the normal view press r again)

l: To change the colour of the light above the radio.

p: To remove the roof of the building.

o: To change the colour of the Skybox.

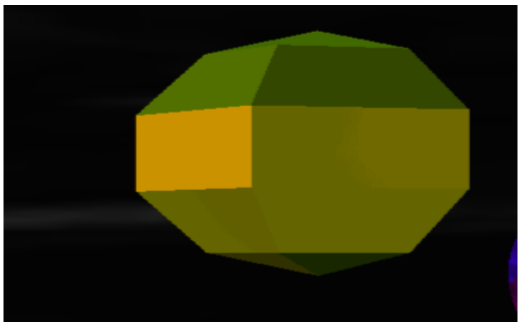
# Scene Explanation Based on Requirements:

## Geometry

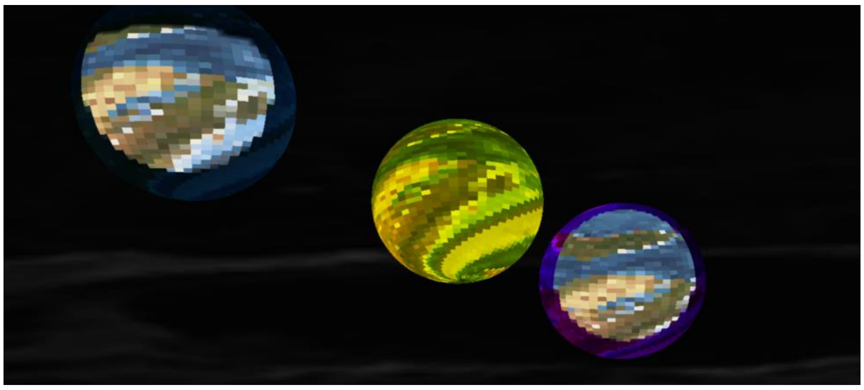
### Procedural generation:

For this project I procedurally create sphere in the "PrecuduallyGeneratedShapes" class in summary what the function does is that it parses the resolution as an input, the resolution determines how spherical the object would be for example if I were to give It a value of 5 it would create a sphere like object that is by no means perfect.

Shape with low resolution:



Shape with higher amount of resolution



Once the resolution is obtained the function modifies it to get the right number of divisions, then calculates the angular increments for the latitude and longitude.

The method iterates through these divisions using 2 loops, moving through individual quads with specified vertices, normal, and texture coordinates to create the spherical object (Note: in this implementation of the sphere the texture coordinates where not calculated properly, but the result is a cool Minecraft like result, so I decided to go with that. In future implementations this would be fixed).

## Skybox:

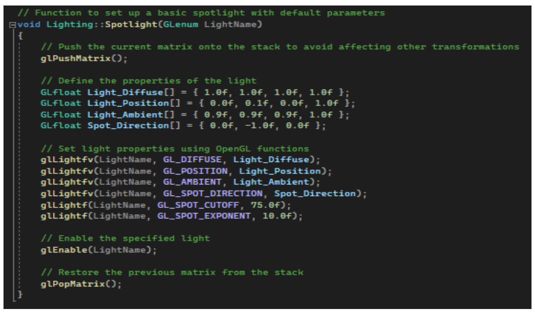
The skybox in this project was created by disable and enabling the depth testing this is to allow the skybox draw around the user but not disrupt, the type of skybox I used was a boxed one as I believe this was the most effective for me, I then made it so that the skybox translation track the cameras position so it's always with the user.

## Hand Created Scene:

The scene in this project was made entirely by hand except for importing assets for use, the room created using simple vertices with uses of scaling to make it rectangular, the stands were also made by hand, all created geometry also have mapped textures and material specifics.

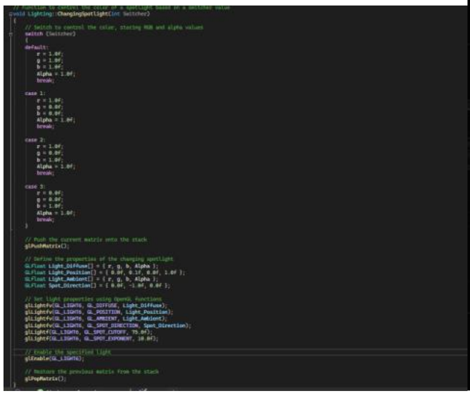
# Lighting

In this project I implemented multiple lights made to achieve different results, the first type of light I created is a spotlight made with the intention to act as an overhead light attached to a lamp to illuminate the object on display below it.
How was this done? This was achieved by making a simple lighting function that takes in a Light Name. I did this so that I do not have to repeat code to make the light every time, when making the light I game it a colour of while and game its ambient properties.

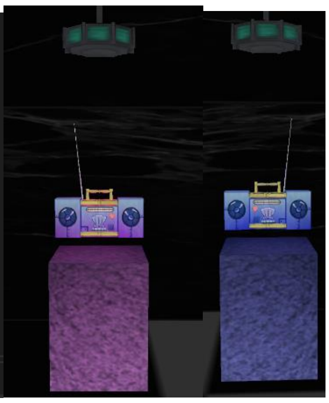


```
// Function to set up a basic spotlight with default parameters
void Lighting::Spotlight(GLenum LightName)
{
    // Push the current matrix onto the stack to avoid affecting other transformations
    glPushMatrix();

    // Define the properties of the light
    GLfloat Light_Diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    GLfloat Light_Position[] = { 0.0f, 0.1f, 0.0f, 1.0f };
    GLfloat Light_Ambient[] = { 0.9f, 0.9f, 0.9f, 1.0f };
    GLfloat Spot_Direction[] = { 0.0f, -1.0f, 0.0f };

    // Set light properties using OpenGL functions
    glLightfv(LightName, GL_DIFFUSE, Light_Diffuse);
    glLightfv(LightName, GL_POSITION, Light_Position);
    glLightfv(LightName, GL_AMBIENT, Light_Ambient);
    glLightfv(LightName, GL_SPOT_DIRECTION, Spot_Direction);
    glLightf(LightName, GL_SPOT_CUTOFF, 75.0f);
    glLightf(LightName, GL_SPOT_EXPONENT, 10.0f);

    // Enable the specified light
    glEnable(LightName);

    // Restore the previous matrix from the stack
    glPopMatrix();
}
```

I also made the user have the ability to change what colour the light is. This is done by making a simple switch statement that stores different RGB and alpha values and when the button L is pressed the colour is changed.

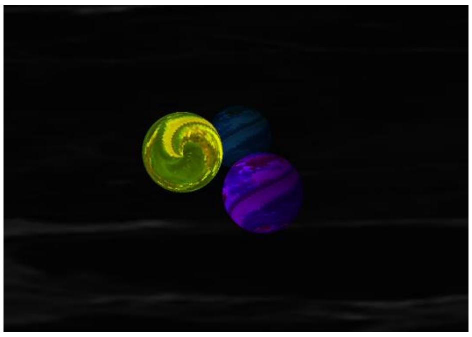


```
// Function to control the color of a spotlight based on a Switcher value
void Lighting::ChangingSpotlight(int Switcher)
{
    // Switch to control the color, storing RGB and alpha values
    switch (Switcher)
    {
    default:
        r = 1.0f;
        g = 1.0f;
        b = 1.0f;
        Alpha = 1.0f;
        break;

    case 1:
        r = 1.0f;
        g = 0.0f;
        b = 0.0f;
        Alpha = 1.0f;
        break;

    case 2:
        r = 1.0f;
        g = 0.0f;
        b = 1.0f;
        Alpha = 1.0f;
        break;

    case 3:
        r = 0.0f;
        g = 0.0f;
        b = 1.0f;
        Alpha = 1.0f;
        break;
    }

    // Push the current matrix onto the stack
    glPushMatrix();

    // Define the properties of the changing spotlight
    GLfloat Light_Diffuse[] = { r, g, b, Alpha };
    GLfloat Light_Position[] = { 0.0f, 0.1f, 0.0f, 1.0f };
    GLfloat Light_Ambient[] = { r, g, b, Alpha };
    GLfloat Spot_Direction[] = { 0.0f, -1.0f, 0.0f };

    // Set light properties using OpenGL functions
    glLightfv(GL_LIGHT6, GL_DIFFUSE, Light_Diffuse);
    glLightfv(GL_LIGHT6, GL_POSITION, Light_Position);
    glLightfv(GL_LIGHT6, GL_AMBIENT, Light_Ambient);
    glLightfv(GL_LIGHT6, GL_SPOT_DIRECTION, Spot_Direction);
    glLightf(GL_LIGHT6, GL_SPOT_CUTOFF, 75.0f);
    glLightf(GL_LIGHT6, GL_SPOT_EXPONENT, 10.0f);

    // Enable the specified light
    glEnable(GL_LIGHT6);

    // Restore the previous matrix from the stack
    glPopMatrix();
}
```

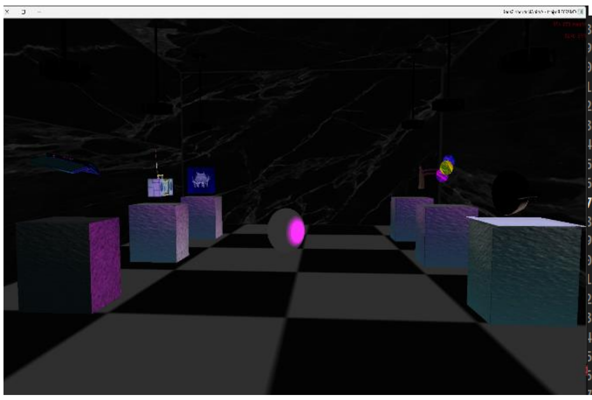


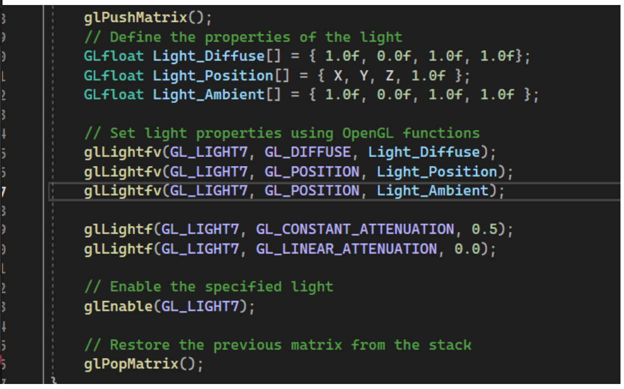
## Materials/Lights

Another aspect in my project that had to do with lighting was materials. I created a separate material class in my program to allow different specifications to be easily changed. A use of the material technique I used was subtle refractions on surfaces, for example in my project I made a small-scale solar system with planets orbiting around a sun. The sun's bright yellow shines its light onto the other planets illuminating their surface.



Another use of the lighting system I have used is the use of a light with both ambient and diffusion properties set to purple to give the room a purple tone. With these various light implementations in my scene, I believe I have effectively utilised different types of lighting that I have learned about.





```
glPushMatrix();
// Define the properties of the light
GLfloat Light_Diffuse[] = { 1.0f, 0.0f, 1.0f, 1.0f};
GLfloat Light_Position[] = { X, Y, Z, 1.0f };
GLfloat Light_Ambient[] = { 1.0f, 0.0f, 1.0f, 1.0f };

// Set light properties using OpenGL functions
glLightfv(GL_LIGHT7, GL_DIFFUSE, Light_Diffuse);
glLightfv(GL_LIGHT7, GL_POSITION, Light_Position);
glLightfv(GL_LIGHT7, GL_POSITION, Light_Ambient);

glLightf(GL_LIGHT7, GL_CONSTANT_ATTENUATION, 0.5);
glLightf(GL_LIGHT7, GL_LINEAR_ATTENUATION, 0.0);

// Enable the specified light
glEnable(GL_LIGHT7);

// Restore the previous matrix from the stack
glPopMatrix();
```

A use of animated lighting is  the a purple light rotating around the shadow giving a glow around the shadow plane

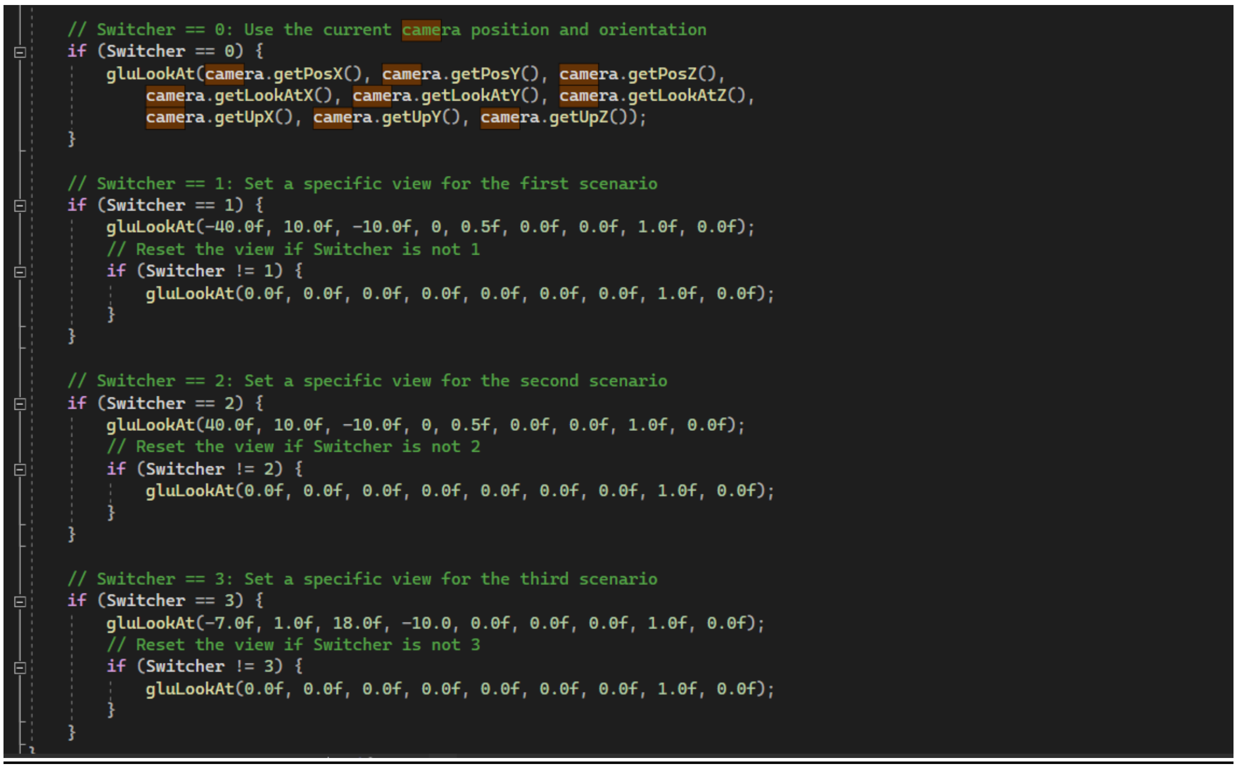
## Camera

### Main Camera:

My project contains only one camera, this camera was made with the help of the video lectures and in person help from lectures. The camera orientation is based on a method that updates the yaw pitch and roll angles as well as a function that allows the user to move the camera either forward backwards, left, right, up and down, the user also has the ability to control the pitch and yaw angle these are the angles of where the camera is looking this is controlled by the mouse.

### Static Camera:

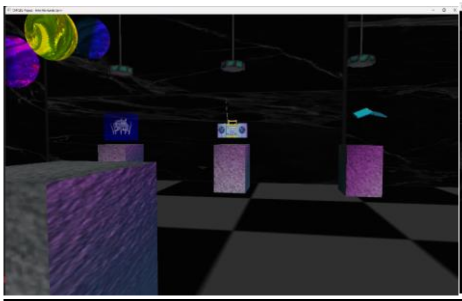
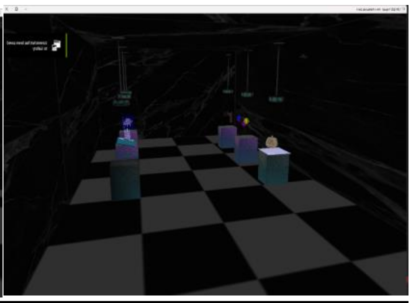
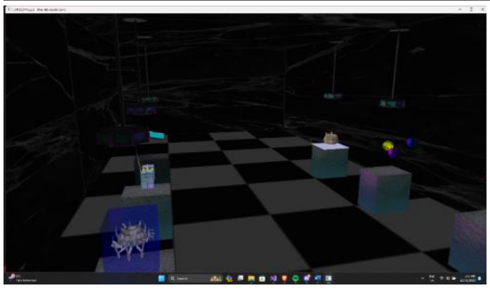
While creating the static cameras I ran into an issue. The issue was that when the player switches to a static camera the angle of the static camera stays with the main camera therefore making the main camera very hard to use and inconvenient for the user. I fixed this issue by creating a simple if statement that switches to a different look at position based on the number attached to the switcher, and if it's no longer on that camera the `gluLookAt` function is set back to 0 on all values so as not to affect the other cameras.



```
// Switcher == 0: Use the current camera position and orientation
if (Switcher == 0) {
    gluLookAt(camera.getPosX(), camera.getPosY(), camera.getPosZ(),
        camera.getLookAtX(), camera.getLookAtY(), camera.getLookAtZ(),
        camera.getUpX(), camera.getUpY(), camera.getUpZ());
}

// Switcher == 1: Set a specific view for the first scenario
if (Switcher == 1) {
    gluLookAt(-40.0f, 10.0f, -10.0f, 0, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f);
    // Reset the view if Switcher is not 1
    if (Switcher != 1) {
        gluLookAt(0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);
    }
}

// Switcher == 2: Set a specific view for the second scenario
if (Switcher == 2) {
    gluLookAt(40.0f, 10.0f, -10.0f, 0, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f);
    // Reset the view if Switcher is not 2
    if (Switcher != 2) {
        gluLookAt(0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);
    }
}

// Switcher == 3: Set a specific view for the third scenario
if (Switcher == 3) {
    gluLookAt(-7.0f, 1.0f, 18.0f, -10.0, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);
    // Reset the view if Switcher is not 3
    if (Switcher != 3) {
        gluLookAt(0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);
    }
}
```

With this fix I was then able to implement static cameras here are all my static cameras

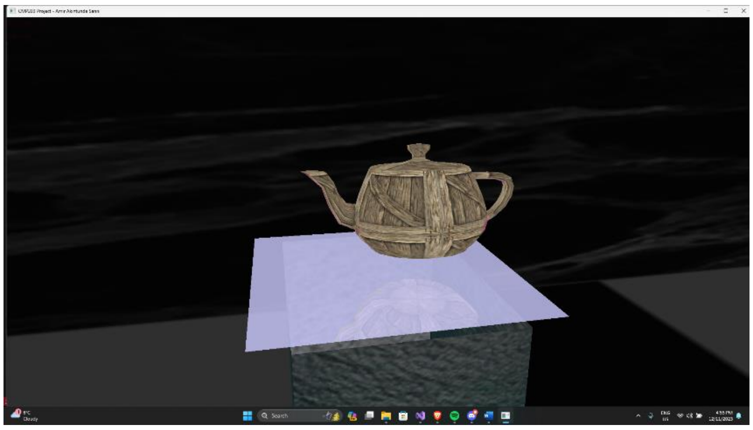
Static Cameras

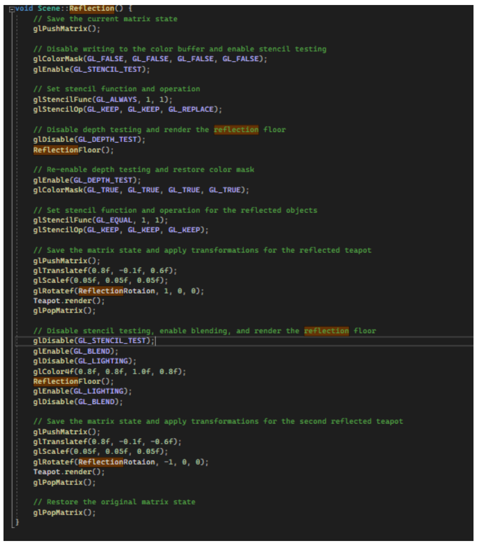
# Advanced Features

## Reflections:

In this project I have implemented a good number of different features including reflections, I was able to implement a technique of reflections where the object in the scene is rendered twice. This is called an imposter mirror, no actual reflecting is taking place, only 2 objects rendered with simple maths applied to it. Here is my implementation of this



How was this done? this was achieved by manipulating the stencil buffer in specific ways



```
void Scene::Reflection() {
    // Save the current matrix state
    glPushMatrix();

    // Disable writing to the color buffer and enable stencil testing
    glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
    glEnable(GL_STENCIL_TEST);

    // Set stencil function and operation
    glStencilFunc(GL_ALWAYS, 1, 1);
    glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);

    // Disable depth testing and render the reflection floor
    glDisable(GL_DEPTH_TEST);
    ReflectionFloor();

    // Re-enable depth testing and restore color mask
    glEnable(GL_DEPTH_TEST);
    glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);

    // Set stencil function and operation for the reflected objects
    glStencilFunc(GL_EQUAL, 1, 1);
    glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

    // Save the matrix state and apply transformations for the reflected teapot
    glPushMatrix();
    glTranslatef(0.8f, -0.1f, 0.6f);
    glScalef(0.05f, 0.05f, 0.05f);
    glRotatef(ReflectionRotaion, 1, 0, 0);
    Teapot.render();
    glPopMatrix();

    // Disable stencil testing, enable blending, and render the reflection floor
    glDisable(GL_STENCIL_TEST);
    glEnable(GL_BLEND);
    glDisable(GL_LIGHTING);
    glColor4f(0.8f, 0.8f, 1.0f, 0.8f);
    ReflectionFloor();
    glEnable(GL_LIGHTING);
    glDisable(GL_BLEND);

    // Save the matrix state and apply transformations for the second reflected teapot
    glPushMatrix();
    glTranslatef(0.8f, -0.1f, -0.6f);
    glScalef(0.05f, 0.05f, 0.05f);
    glRotatef(ReflectionRotaion, -1, 0, 0);
    Teapot.render();
    glPopMatrix();

    // Restore the original matrix state
    glPopMatrix();
}
```

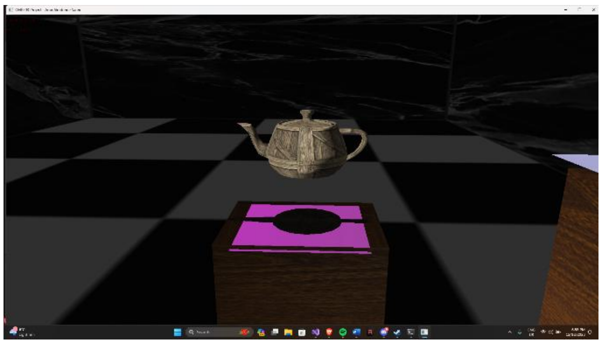
In this code the first thing we do is use the glpushmatix so that any code written before the function does not affect the reflection after that we disable colour writing and enable stencil testing next the stencil function is set to always pass and the stencil value is set to one after the depth testing the disabled and the floor is rendered this is the reflective surface.

The next step in this process is to re-enable depth testing to restore the colour mask and then draw the teapot. The last few steps would be to disable Stencil Testing, Enable Blending, and Render Reflection Floor Again and lastly render the model.

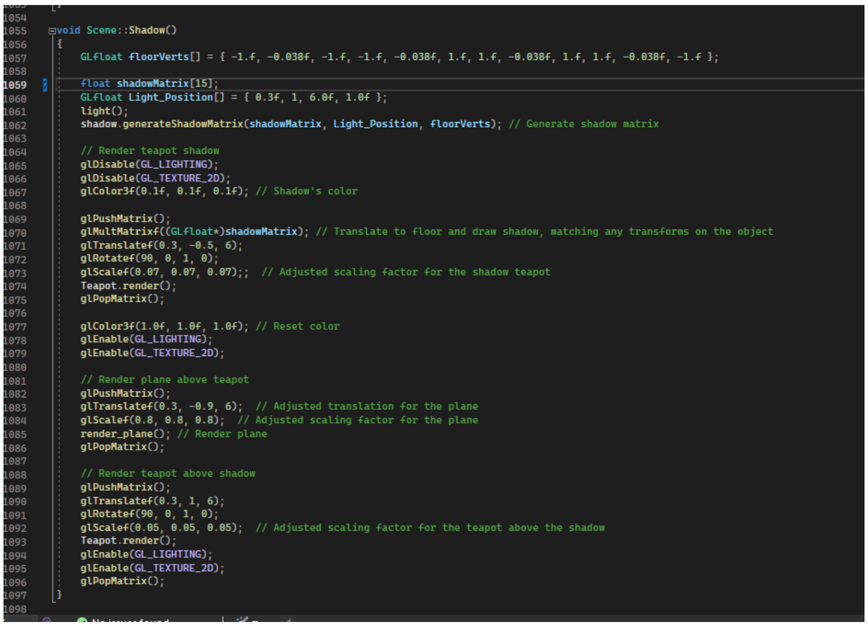
## Shadow

In this project I also implemented shadows, the type of shadow I used in the project is called a planar shadow, this types of shadows are created by building a special matrix transforms that flattens an object's geometry into a flat plane when It is rendered, I chose this type of shadow because it was one of the best looking and is not overly complex to implement, the only downside about this type of shadow is that it only looks good on flat surfaces which limited its use case.

Here is my visual implementation:



here is my code implementation:



```cpp
void Scene::Shadow()
{
    GLfloat floorVerts[] = { -1.f, -0.038f, -1.f, -1.f, -0.038f, 1.f, 1.f, -0.038f, 1.f, 1.f, -0.038f, -1.f };

    float shadowMatrix[15];
    GLfloat Light_Position[] = { 0.3f, 1, 6.0f, 1.0f };
    light();
    shadow.generateShadowMatrix(shadowMatrix, Light_Position, floorVerts); // Generate shadow matrix

    // Render teapot shadow
    glDisable(GL_LIGHTING);
    glDisable(GL_TEXTURE_2D);
    glColor3f(0.1f, 0.1f, 0.1f); // Shadow's color

    glPushMatrix();
    glMultMatrixf((GLfloat*)shadowMatrix); // Translate to floor and draw shadow, matching any transforms on the object
    glTranslatef(0.3, -0.5, 6);
    glRotatef(90, 0, 1, 0);
    glScalef(0.07, 0.07, 0.07);;  // Adjusted scaling factor for the shadow teapot
    Teapot.render();
    glPopMatrix();

    glColor3f(1.0f, 1.0f, 1.0f); // Reset color
    glEnable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);

    // Render plane above teapot
    glPushMatrix();
    glTranslatef(0.3, -0.9, 6);  // Adjusted translation for the plane
    glScalef(0.8, 0.8, 0.8);  // Adjusted scaling factor for the plane
    render_plane(); // Render plane
    glPopMatrix();

    // Render teapot above shadow
    glPushMatrix();
    glTranslatef(0.3, 1, 6);
    glRotatef(90, 0, 1, 0);
    glScalef(0.05, 0.05, 0.05);  // Adjusted scaling factor for the teapot above the shadow
    Teapot.render();
    glEnable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);
    glPopMatrix();
}
```

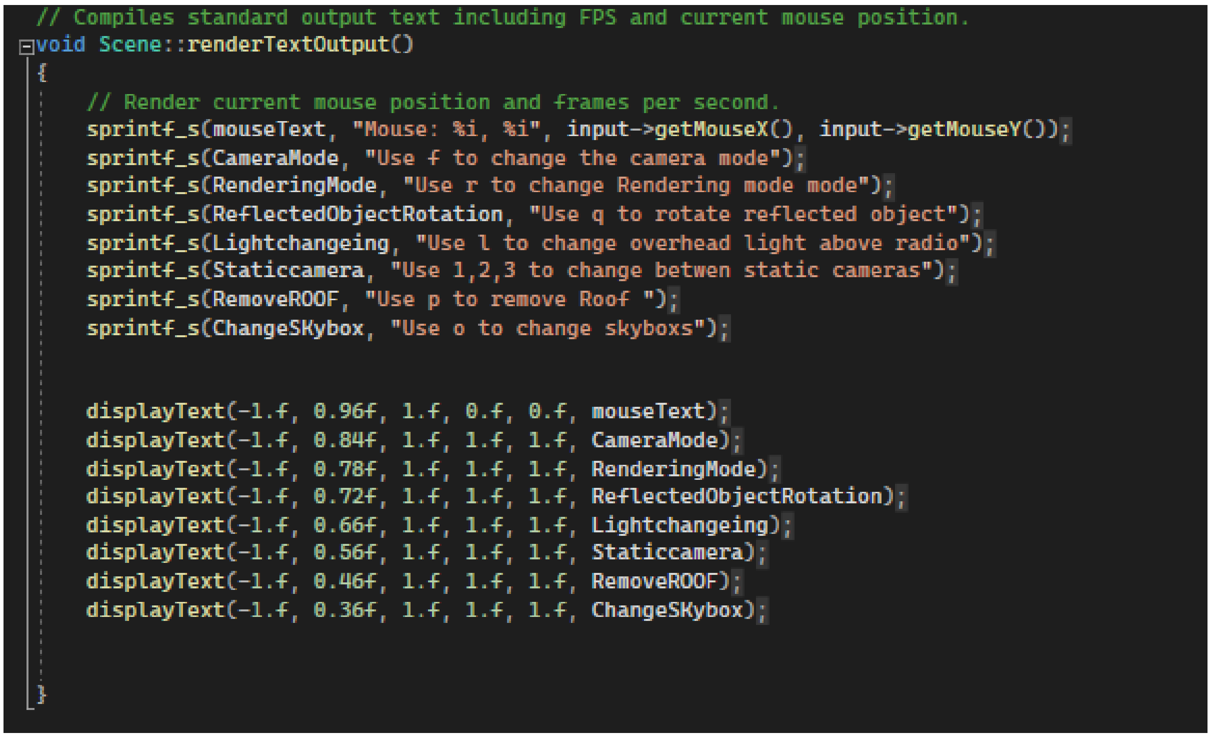
The shadow in my scene is generated by using a shadow matrix based on the specified light position I created and floor specific vertices. The rendering process involves the use of multiple OpenGL functions and matrix transformation.

Firstly, the floor vertices are created, and a shadow matrix is generated using the shadow.generateShadowMatrix function (code provided by lecturer). The next rendering of the teapot shadow involves disabling textures and lighting, setting a dark colour for the shadow in my case I chose black, and applying different transformations including translation, scaling, and rotation based on the shadow matrix. This makes sure the shadow aligns with the teapot's position and orientation.

The last few things I do is re-enables lighting and textures and render another plane and adjusting its translation and scaling. Lastly, another teapot is rendered above the shadow, applying transformations and scaling factors, all these steps leading to a working shadow in my scene.

## Usability

Whilst making this project i created things in a way to make everything easy to see and easy to understand, I believe I achieved this goal as every technique learned is on display and does not have to be found, I also added instructions so that the user is able to know what to do, implementing instructions was very easy here is how I did this.



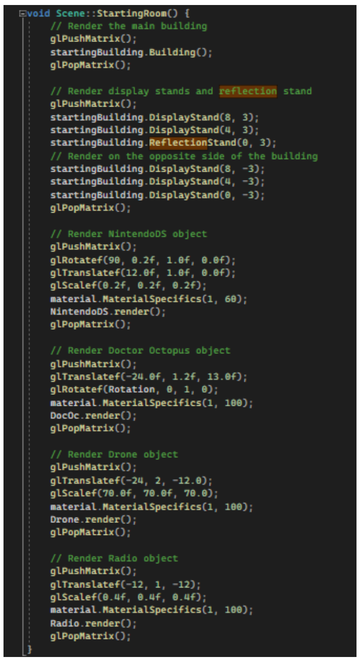
```
// Compiles standard output text including FPS and current mouse position.
void Scene::renderTextOutput()
{
    // Render current mouse position and frames per second.
    sprintf_s(mouseText, "Mouse: %i, %i", input->getMouseX(), input->getMouseY());
    sprintf_s(CameraMode, "Use f to change the camera mode");
    sprintf_s(RenderingMode, "Use r to change Rendering mode mode");
    sprintf_s(ReflectedObjectRotation, "Use q to rotate reflected object");
    sprintf_s(Lightchangeing, "Use l to change overhead light above radio");
    sprintf_s(Staticcamera, "Use 1,2,3 to change betwen static cameras");
    sprintf_s(RemoveROOF, "Use p to remove Roof ");
    sprintf_s(ChangeSKybox, "Use o to change skyboxs");


    displayText(-1.f, 0.96f, 1.f, 0.f, 0.f, mouseText);
    displayText(-1.f, 0.84f, 1.f, 1.f, 1.f, CameraMode);
    displayText(-1.f, 0.78f, 1.f, 1.f, 1.f, RenderingMode);
    displayText(-1.f, 0.72f, 1.f, 1.f, 1.f, ReflectedObjectRotation);
    displayText(-1.f, 0.66f, 1.f, 1.f, 1.f, Lightchangeing);
    displayText(-1.f, 0.56f, 1.f, 1.f, 1.f, Staticcamera);
    displayText(-1.f, 0.46f, 1.f, 1.f, 1.f, RemoveROOF);
    displayText(-1.f, 0.36f, 1.f, 1.f, 1.f, ChangeSKybox);



}
```

In this code I am using the sprint_s function what stores the information of the char then it is sent to display text which displays in in set coordinates on the screen

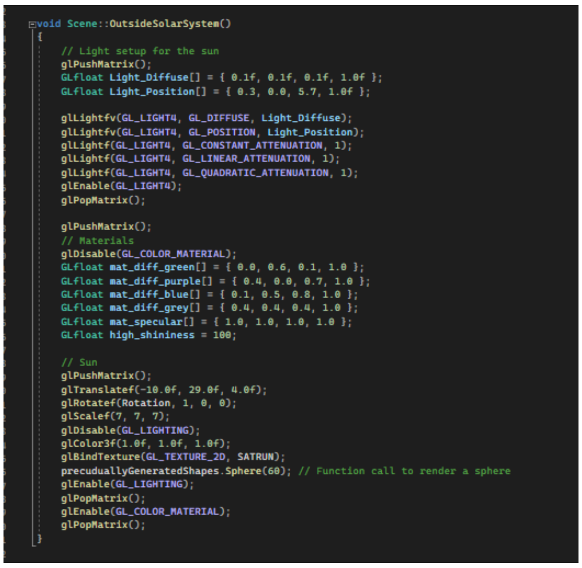
# Code Structure

## Hierarchical modelling:

Throughout my project I have made use of hierarchical modelling using the glpushmatrix and glpopmatrixi to make my project easy to understand and to prevent code written for other functions interfere with other lines of code, here are some examples of my code:



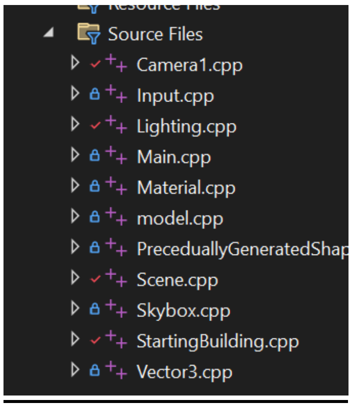
```cpp
void Scene::StartingRoom() {
    // Render the main building
    glPushMatrix();
    startingBuilding.Building();
    glPopMatrix();

    // Render display stands and reflection stand
    glPushMatrix();
    startingBuilding.DisplayStand(8, 3);
    startingBuilding.DisplayStand(4, 3);
    startingBuilding.ReflectionStand(0, 3);
    // Render on the opposite side of the building
    startingBuilding.DisplayStand(8, -3);
    startingBuilding.DisplayStand(4, -3);
    startingBuilding.DisplayStand(0, -3);
    glPopMatrix();

    // Render NintendoDS object
    glPushMatrix();
    glRotatef(90, 0.2f, 1.0f, 0.0f);
    glTranslatef(12.0f, 1.0f, 0.0f);
    glScalef(0.2f, 0.2f, 0.2f);
    material.MaterialSpecifics(1, 60);
    NintendoDS.render();
    glPopMatrix();

    // Render Doctor Octopus object
    glPushMatrix();
    glTranslatef(-24.0f, 1.2f, 13.0f);
    glRotatef(Rotation, 0, 1, 0);
    material.MaterialSpecifics(1, 100);
    DocOc.render();
    glPopMatrix();

    // Render Drone object
    glPushMatrix();
    glTranslatef(-24, 2, -12.0);
    glScalef(70.0f, 70.0f, 70.0);
    material.MaterialSpecifics(1, 100);
    Drone.render();
    glPopMatrix();

    // Render Radio object
    glPushMatrix();
    glTranslatef(-12, 1, -12);
    glScalef(0.4f, 0.4f, 0.4f);
    material.MaterialSpecifics(1, 100);
    Radio.render();
    glPopMatrix();
}
```



```cpp
void Scene::OutsideSolarSystem()
{
    // Light setup for the sun
    glPushMatrix();
    GLfloat Light_Diffuse[] = { 0.1f, 0.1f, 0.1f, 1.0f };
    GLfloat Light_Position[] = { 0.3, 0.0, 5.7, 1.0f };

    glLightfv(GL_LIGHT4, GL_DIFFUSE, Light_Diffuse);
    glLightfv(GL_LIGHT4, GL_POSITION, Light_Position);
    glLightf(GL_LIGHT4, GL_CONSTANT_ATTENUATION, 1);
    glLightf(GL_LIGHT4, GL_LINEAR_ATTENUATION, 1);
    glLightf(GL_LIGHT4, GL_QUADRATIC_ATTENUATION, 1);
    glEnable(GL_LIGHT4);
    glPopMatrix();

    glPushMatrix();
    // Materials
    glDisable(GL_COLOR_MATERIAL);
    GLfloat mat_diff_green[] = { 0.0, 0.6, 0.1, 1.0 };
    GLfloat mat_diff_purple[] = { 0.4, 0.0, 0.7, 1.0 };
    GLfloat mat_diff_blue[] = { 0.1, 0.5, 0.8, 1.0 };
    GLfloat mat_diff_grey[] = { 0.4, 0.4, 0.4, 1.0 };
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat high_shininess = 100;

    // Sun
    glPushMatrix();
    glTranslatef(-10.0f, 29.0f, 4.0f);
    glRotatef(Rotation, 1, 0, 0);
    glScalef(7, 7, 7);
    glDisable(GL_LIGHTING);
    glColor3f(1.0f, 1.0f, 1.0f);
    glBindTexture(GL_TEXTURE_2D, SATRUN);
    precuduallyGeneratedShapes.Sphere(60); // Function call to render a sphere
    glEnable(GL_LIGHTING);
    glPopMatrix();
    glEnable(GL_COLOR_MATERIAL);
    glPopMatrix();
}
```

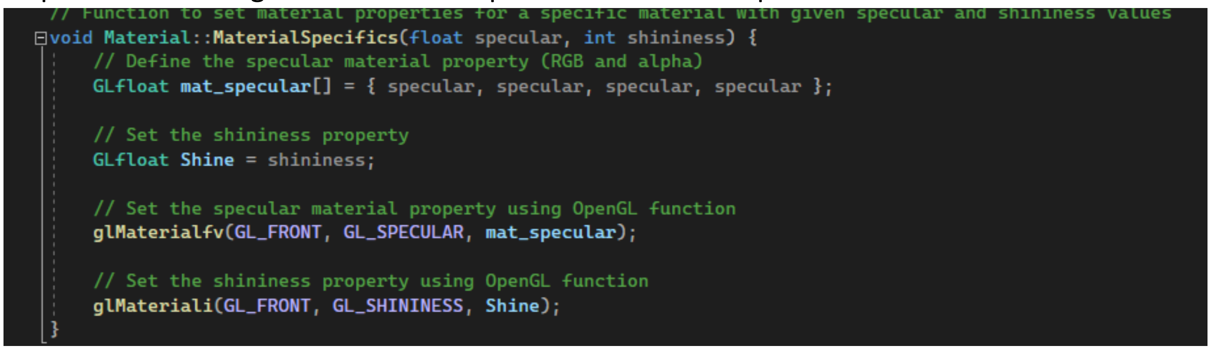
## General readability and reusability:

Throughout this project I have used many object-oriented techniques such as my use of many functions and classes, while making this project I made multiple classes to space out my code.



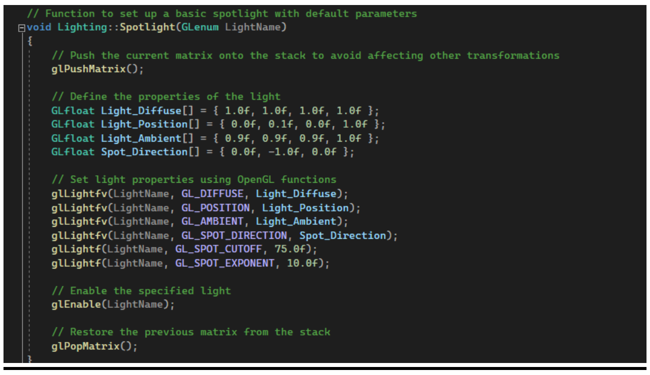
```
Source Files
    Camera1.cpp
    Input.cpp
    Lighting.cpp
    Main.cpp
    Material.cpp
    model.cpp
    PrecedullyGeneratedShap
    Scene.cpp
    Skybox.cpp
    StartingBuilding.cpp
    Vector3.cpp
```

I also made my code as readable as possible However, I ran into an issue where making lights and became confusing, hard to read, and cumbersome to navigate., so I decided to make them into separate functions in their respective classes and making their requirements changeable from the inputs here is an example of this:



```
// Function to set material properties for a specific material with given specular and shininess values
void Material::MaterialSpecifics(float specular, int shininess) {
    // Define the specular material property (RGB and alpha)
    GLfloat mat_specular[] = { specular, specular, specular, specular };

    // Set the shininess property
    GLfloat Shine = shininess;

    // Set the specular material property using OpenGL function
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);

    // Set the shininess property using OpenGL function
    glMateriali(GL_FRONT, GL_SHININESS, Shine);
}
```

This approach significantly reduced redundancy, saving me at least 90 lines of code. I applied a similar strategy in my light class, streamlining the code and enhancing its maintainability.



```
// Function to set up a basic spotlight with default parameters
void Lighting::Spotlight(GLenum LightName)
{
    // Push the current matrix onto the stack to avoid affecting other transformations
    glPushMatrix();

    // Define the properties of the light
    GLfloat Light_Diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    GLfloat Light_Position[] = { 0.0f, 0.1f, 0.0f, 1.0f };
    GLfloat Light_Ambient[] = { 0.9f, 0.9f, 0.9f, 1.0f };
    GLfloat Spot_Direction[] = { 0.0f, -1.0f, 0.0f };

    // Set light properties using OpenGL functions
    glLightfv(LightName, GL_DIFFUSE, Light_Diffuse);
    glLightfv(LightName, GL_POSITION, Light_Position);
    glLightfv(LightName, GL_AMBIENT, Light_Ambient);
    glLightfv(LightName, GL_SPOT_DIRECTION, Spot_Direction);
    glLightf(LightName, GL_SPOT_CUTOFF, 75.0f);
    glLightf(LightName, GL_SPOT_EXPONENT, 10.0f);

    // Enable the specified light
    glEnable(LightName);

    // Restore the previous matrix from the stack
    glPopMatrix();
}
```

## Room for improvement

Although I am proud of my work for this module, I believe there are various ways I could improve this project. Firstly, by creating assets tailor made for this project so that all the assets follow a certain aesthetic, whereas in my current implementation of the project I got all assets and textures from free online sources which does not always match my vision for the project.

Another aspect I would like to improve on this project would be in my code, I feel I could have taken different approaches to some aspects of my code. I sometimes have functions that are created for testing but are not actually used and sometimes repeat code leading my code in the Scene reaching to over 1000 lines of code.

Another part for improvement I would want to make in this project is my use of the 3d audio, I initially wanted to make it so that the audio comes from the radio, but I was not able to get that working properly, this would have increased the immersion into the scene.

lastly, I believe I could have made better use of the lighting system in open gl, the way the lighting in my scene is set up not is okay but I would like to have add more better looking lights allowing them to complement each other more effectively, sometimes in my project light does not even reach specific parts of the game but reaches other outside the starting building.

## Conclusion

In conclusion I believe my project meets the requirements provided in the brief, providing the user with a visually appealing scene that incorporates different techniques such as shadows and reflections with the use of the stencil buffer, a usable camera, procedurally generated shapes and more.

The project demonstrates my commitment to easy usability by making sure that all implemented techniques are displayed and easily accessible, removing the need for the users to actively search for specific features. Clear instructions are also displayed at all times on screen and further enhance the users experience, leading to a more seamless user experience.

My want for easy usability is also seen in my code with the implementation of clear functions and reasonably commented code.

The code structure also follows the use of good practices, including the use of hierarchical modelling with glPushMatrix and glPopMatrix to enhance code visibility and prevent interference between different code sections. Object-oriented programming principles are also used throughout the project, with the use of multiple classes and functions to improve readability and reusability.

Despite encountering challenges, such as the implementation of different types of lights and light sources, the project's design prioritises clarity and ease of navigation.

<u>References:</u>

Treyarch (2004) Spider-Man 2 [Video game]. Sony Interactive Entertainment https://www.textures-resource.com/gamecube/spiderman2/texture/21428/?source=genre

ARCHITONIC. (N/A) ARCHITONIC. Available at: https://www.architonic.com/en/product/gani-marble-tiles-black-nero-marquina/1485967

(Accessed: 12/ December 2023)

Shutterstock. (N/A) Shutterstock. Available at: https://www.shutterstock.com/image-photo/black-white-checkered-floor-tiles-background-1020935170

(Accessed: 12/ December 2023)

Heavy Iron Studios (2004) The SpongeBob SquarePants Movie [Video game]. AWE Games https://www.models-resource.com/pc_computer/thespongebobsquarepantsmovie/model/22983/

planetary visions. (2007) online Library. Available at: https://www.flickr.com/photos/42084302@N05/3884071286

(Accessed: 12/ December 2023)

rolffimages. (N/A) Adobe Stock. Available at: https://stock.adobe.com/uk/search?k=moon+surface+texture&asset_id=235976501 (Accessed: 11/ December 2023)

2K Games (2007) BioShock [Video game]. 2K Games : https://www.models-resource.com/pc_computer/bioshock/model/24022/

Leszek Maziarz. (N/A ) Adobe Stock. Available at : https://stock.adobe.com/uk/images/fine-old-oak-wood-grain-texture/73959889 (Accessed: 12/ December 2023)

Sweetydu972M (2009) Chet Baker - Almost blue. Available at:
https://www.youtube.com/watch?v=z4PKzz81m5c