**Description:**

In this assignment, a 3D graphics application was developed to create and interact with objects in a virtual environment. The following components were implemented:

1. **3D Models**: Basic 3D models, such as table, chair, refrigerator, fan, etc. were created using geometric primitives like cubes. These primitives were combined and modified to form more complex shapes, representing the objects in the scene.
2. **Transformation System**: A transformation system was developed to manipulate the 3D objects by applying translation, rotation, and scaling operations. This system allowed for changes in the position, orientation, and size of the objects within the 3D space.
3. **User Interaction**: User controls were implemented to enable interaction with the objects. Through keyboard and mouse inputs, the user can move, rotate, and resize objects, as well as navigate through the scene. This added an interactive aspect to the 3D environment.
4. **Camera System**: A dynamic camera system was integrated, allowing for changes in the perspective of the scene. This system enabled zooming and rotating, offering an immersive experience by letting users explore the scene from different angles.

These components together formed a comprehensive 3D environment where users could interact with objects, explore from various angles, and manipulate the objects in real-time. The integration of these elements resulted in a dynamic, visually engaging 3D scene.

**Program and Results:**

**// Bird’s EyeView**

glm::mat4 view;

if (birdEyeView) {

glm::vec3 birdEyePosition(0.0f, 10.0f, 0.0f);

glm::vec3 birdEyeTarget(0.0f, 0.0f, 0.0f);

glm::vec3 upVector(0.0f, 0.0f, 1.0f);

view = customLookAt(birdEyePosition, birdEyeTarget, upVector);

}

else {

view = basic\_camera.createViewMatrix();

}

if (glfwGetKey(window, GLFW\_KEY\_B) == GLFW\_PRESS) {

birdEyeView = true;

}

if (glfwGetKey(window, GLFW\_KEY\_N) == GLFW\_PRESS) {

birdEyeView = false;

}



**// Rotation Matrices**

// main.cpp

glm::mat4 RotationMatricesY(float theta) {

float cosTheta = cos(theta);

float sinTheta = sin(theta);

return glm::mat4(

cosTheta, 0.0f, sinTheta, 0.0f,

0.0f, 1.0f, 0.0f, 0.0f,

-sinTheta, 0.0f, cosTheta, 0.0f,

0.0f, 0.0f, 0.0f, 1.0f );

}

glm::mat4 RotationMatricesZ(float theta) {

float cosTheta = cos(theta);

float sinTheta = sin(theta);

return glm::mat4(

cosTheta, -sinTheta, 0.0f, 0.0f,

sinTheta, cosTheta, 0.0f, 0.0f,

0.0f, 0.0f, 1.0f, 0.0f,

0.0f, 0.0f, 0.0f, 1.0f);

}

|  |  |
| --- | --- |
|  |  |
| Rotation About X-axis | Rotation About Z-axis |
|  | |
| Rotation About Y-axis | |

**// Moving**

if (glfwGetKey(window, GLFW\_KEY\_UP) == GLFW\_PRESS) translate\_Y += 0.1;

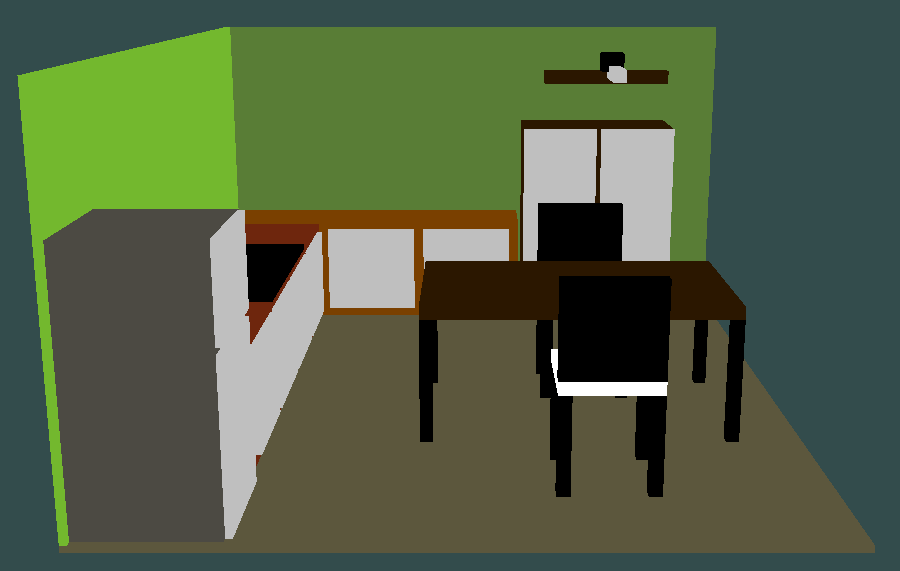
if (glfwGetKey(window, GLFW\_KEY\_DOWN) == GLFW\_PRESS) translate\_Y -= 0.1;

if (glfwGetKey(window, GLFW\_KEY\_LEFT) == GLFW\_PRESS) translate\_X += 0.1;

if (glfwGetKey(window, GLFW\_KEY\_RIGHT) == GLFW\_PRESS) translate\_X -= 0.1;

if (glfwGetKey(window, GLFW\_KEY\_MINUS) == GLFW\_PRESS) translate\_Z += 0.1;

if (glfwGetKey(window, GLFW\_KEY\_EQUAL) == GLFW\_PRESS) translate\_Z -= 0.1;



**// Rotating the camera around a look at point using key F**

glm::vec3 lookAtPoint = glm::vec3(0.0f, 0.0f, 0.0f);

float radius = 10.0f, angle = 0.0f; // Current angle for rotation

void processInput(GLFWwindow\* window) {

if (glfwGetKey(window, GLFW\_KEY\_F) == GLFW\_PRESS) {

angle += 50.0f \* deltaTime; // Increment angle (speed adjusted by deltaTime)

if (angle >= 360.0f) angle -= 360.0f; // Keep angle within [0, 360] range

}

}

glm::mat4 rotateCameraAroundPoint() {

float cameraX = lookAtPoint.x + radius \* cos(glm::radians(angle));

float cameraZ = lookAtPoint.z + radius \* sin(glm::radians(angle));

glm::vec3 cameraPosition = glm::vec3(cameraX, 0.0f, cameraZ); // New position

return customLookAt(cameraPosition, lookAtPoint, glm::vec3(0.0f, 1.0f, 0.0f));

}

glm::mat4 view;

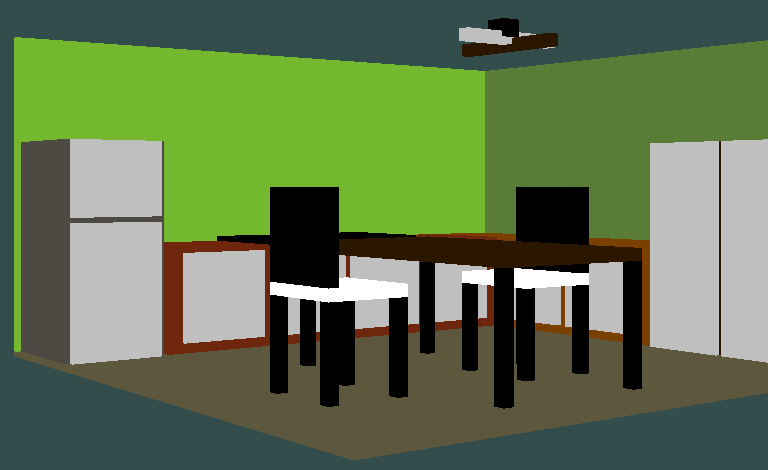
if (rotatingCamera) {

view = rotateCameraAroundPoint();

} else {

view = basic\_camera.createViewMatrix();

}



**// Rotating Fan**

if (fanOn) {

translateMatrix = glm::translate(identityMatrix, glm::vec3(0.5, 2.70f, 1.0f));

scaleMatrix = glm::scale(identityMatrix, glm::vec3(2.0f, 0.2f, 0.2f));

rotateYMatrix = RotationMatricesY(glm::radians(r\_fan));

//model = translateMatrix \* rotateYMatrix \* scaleMatrix;

model = translateMatrix \* glm::translate(identityMatrix, glm::vec3(0.5f, 0.0, 0.0f)) \* rotateYMatrix \* glm::translate(identityMatrix, glm::vec3(-0.5, 0.0, 0.0f)) \* scaleMatrix;

ourShader.setMat4("model", model);

ourShader.setVec4("color", glm::vec4(0.17f, 0.09f, 0.00f, 1.0f));

glBindVertexArray(VAO);

glDrawElements(GL\_TRIANGLES, 36, GL\_UNSIGNED\_INT, 0);

// Blade 1

translateMatrix = glm::translate(identityMatrix, glm::vec3(0.5, 2.75f, 1.0f));

scaleMatrix = glm::scale(identityMatrix, glm::vec3(2.0f, 0.2f, 0.2f));

rotateYMatrix = RotationMatricesY(glm::radians(r\_fan + 90));

//model = translateMatrix \* rotateYMatrix \* scaleMatrix;

model = translateMatrix \* glm::translate(identityMatrix, glm::vec3(0.5f, 0.0, 0.0f)) \* rotateYMatrix \* glm::translate(identityMatrix, glm::vec3(-0.5, 0.0, 0.0f)) \* scaleMatrix;

ourShader.setMat4("model", model);

ourShader.setVec4("color", glm::vec4(0.75, 0.75, 0.75, 1));

r\_fan += 20;

}

else {

translateMatrix = glm::translate(identityMatrix, glm::vec3(0.5, 2.70f, 1.0f));

scaleMatrix = glm::scale(identityMatrix, glm::vec3(2.0f, 0.2f, 0.2f));

rotateYMatrix = RotationMatricesY(glm::radians(r\_fan));

//model = translateMatrix \* rotateYMatrix \* scaleMatrix;

model = translateMatrix \* glm::translate(identityMatrix, glm::vec3(0.5f, 0.0, 0.0f)) \* rotateYMatrix \* glm::translate(identityMatrix, glm::vec3(-0.5, 0.0, 0.0f)) \* scaleMatrix;

ourShader.setMat4("model", model);

ourShader.setVec4("color", glm::vec4(0.17f, 0.09f, 0.00f, 1.0f));

glBindVertexArray(VAO);

glDrawElements(GL\_TRIANGLES, 36, GL\_UNSIGNED\_INT, 0);

// Blade 2

translateMatrix = glm::translate(identityMatrix, glm::vec3(0.5, 2.75f, 1.0f));

scaleMatrix = glm::scale(identityMatrix, glm::vec3(2.0f, 0.2f, 0.2f));

rotateYMatrix = RotationMatricesY(glm::radians(r\_fan + 90));

//model = translateMatrix \* rotateYMatrix \* scaleMatrix;

model = translateMatrix \* glm::translate(identityMatrix, glm::vec3(0.5f, 0.0, 0.0f)) \* rotateYMatrix \* glm::translate(identityMatrix, glm::vec3(-0.5, 0.0, 0.0f)) \* scaleMatrix;

ourShader.setMat4("model", model);

ourShader.setVec4("color", glm::vec4(0.75, 0.75, 0.75, 1));

}

glBindVertexArray(VAO);

glDrawElements(GL\_TRIANGLES, 36, GL\_UNSIGNED\_INT, 0);

**// Customized look-at function**

glm::mat4 customLookAt(glm::vec3 eye, glm::vec3 center, glm::vec3 up) {

// Calculate the direction vectors

glm::vec3 zAxis = glm::normalize(eye - center); // Forward vector

glm::vec3 xAxis = glm::normalize(glm::cross(glm::normalize(up), zAxis)); // Right vector

glm::vec3 yAxis = glm::cross(zAxis, xAxis); // Up vector

glm::mat4 view(1.0f);

view[0][0] = xAxis.x;

view[1][0] = xAxis.y;

view[2][0] = xAxis.z;

view[0][1] = yAxis.x;

view[1][1] = yAxis.y;

view[2][1] = yAxis.z;

view[0][2] = zAxis.x;

view[1][2] = zAxis.y;

view[2][2] = zAxis.z;

view[3][0] = -glm::dot(xAxis, eye);

view[3][1] = -glm::dot(yAxis, eye);

view[3][2] = -glm::dot(zAxis, eye);

return view;

}