Desmos From Temu

Dimitri

today

1 Introduction

This project implements a 3D visualization application using. Make geometric shapes such as cubes, pyramids, spheres, ellipses, and toruses, and allow interaction.

2 Core Components

2.1 Event Handling and User Interaction

The `controls.h` not really deep. control logic.

```
void handleMouseWheelScrolled(Event event, float &zoom) {
    zoom += event.mouseWheelScroll.delta * 10.0f;
    if (zoom < 10.0f) {
        zoom = 10.0f;
    }
}
```

This function updates the zoom factor by adjusting it based on the scroll wheel delta. The zoom value is constrained to a minimum of 10.0 to prevent excessive zooming out.

2.2 Linear Algebra for 3D Graphics

The `linear_algebra.h` file provides classes for 3D vector and matrix operations(explained in other thing)

- Vector Operations: The `Vec3` class handles basic vector arithmetic:

```
Vec3 operator+(const Vec3& other) const {
    return Vec3(x + other.x, y + other.y, z + other.z);
}
```

- Matrix Operation: The `Mat3` class includes a method for creating a rotation matrix:

```
static Mat3 rotationMatrix(float angle, const Vec3& axis) {
    // compute and do the the rotation matrix
}
```

this is how it can do 3d stuff like spin

2.3 Shape Generation

The `shapes.h` file defines functions to create vertices and edges for geometric shapes. For instance, generating a cube:

```
void createCube(vector<Vec3> &cubeVertices, vector<pair<int, int>> &cubeEdges, Vec3 center = Vec3(0,
    cubeVertices = {
        // Define cube vertices
    };
```

```
    cubeEdges = {
        // Define cube edges
    };
}
```

THIS CAN BE SCALED TO ANYTHING ELSE

2.4 Rendering and Projection

The `main` function orchestrates rendering:

```
while (window.isOpen()) {
    // Handle events
    // Apply transformations
    // Project points
    // Draw shapes
    window.display();
}
```

Projection: To project 3D points onto a 2D screen, we use a perspective projection formula:

```
Vector2f project(const Vec3& point, float zoom) {
    float scale = zoom / (point.z + 300.0f);
    return Vector2f(point.x * scale + WINDOW_WIDTH / 2, -point.y * scale + WINDOW_HEIGHT / 2);
}
```

The `project` function converts a 3D point into 2D screen coordinates by applying a perspective transformation that scales the point based on its depth (`z-coordinate`).

THIS scaling factor, which is changed/influenced by the zoom level and depth, makes sure that points further from the camera appear smaller so like if im standing farther away i look shorter.

THEN, the function adjusts the coordinates to center them on the screen, translating the 3D perspective into a 2D display.

3 Mathematical and Logical Stuff

3.1 Matrix Transformations

In 3d graphics matrices are basically all you need for this. Do i funny understand all of how it works? no.

- Moving objects in space. - Rotating objects around an axis. The rotation matrix for a rotation by angle θ around a unit axis vector $\mathbf{a} = (a_x, a_y, a_z)$ is:

$$R = \begin{bmatrix} \cos\theta + a_x^2(1-\cos\theta) & a_x a_y(1-\cos\theta) - a_z\sin\theta & a_x a_z(1-\cos\theta) + a_y\sin\theta \\ a_y a_x(1-\cos\theta) + a_z\sin\theta & \cos\theta + a_y^2(1-\cos\theta) & a_y a_z(1-\cos\theta) - a_x\sin\theta \\ a_z a_x(1-\cos\theta) - a_y\sin\theta & a_z a_y(1-\cos\theta) + a_x\sin\theta & \cos\theta + a_z^2(1-\cos\theta) \end{bmatrix} \tag{1}$$

- This is called the rodrigues rotation formula

3.2 Projection

The perspective projection formula maps 3D coordinates to 2D screen coordinates:

$$x_{\text{screen}} = \frac{x_{\text{world}} \cdot z_{\text{screen}}}{z_{\text{world}} + d} \tag{2}$$

$$y_{\text{screen}} = \frac{y_{\text{world}} \cdot z_{\text{screen}}}{z_{\text{world}} + d} \tag{3}$$

Here, d represents the distance from the camera to the projection plane, and z_{screen} is a factor for scaling based on the zoom level.

4 Conclusion

Conclude