Computer Graphics

(UCS505)

Project on

3D Smart Classroom

Submitted By

Arjun Chouksey	102203044
Raunak Shahi	102203054
Dev Mehta	102203073

3C61

B.E. Third Year – COE

Submitted To:

Dr. Vaishali Kansal

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Jan-May 2025

TABLE OF CONTENTS

Sr. No.	**Description**	**Page No.**
1.	Introduction to Project	1-3
2	User Defined Functions	4-9
3.	Source Code	10-29
4.	Screenshots	30-37
5.	References	38

INTRODUCTION

1. Project Overview-

Smart classrooms are technologically enhanced learning environments that integrate various digital tools and resources to improve the teaching and learning experience. These classrooms leverage technology to create dynamic, interactive, and engaging educational spaces.

Our project leverages OpenGL to render a 3D Smart Classroom Environment. It constructs the room geometry with polygons and applies textures to simulate furniture, screens, and other objects. The program implements a virtual camera that users can manipulate to explore the scene. Furthermore, it incorporates lighting models to create a realistic feel, allowing for adjustments to ambient light and the positioning of virtual spotlights.

By culminating this 3D Classroom project, our goal was to demonstrate the power of Computer Graphics in crafting immersive and interactive learning environments.
Through this project, we aimed to showcase the potential of Computer Graphics in revolutionizing classroom experiences, allowing educators to create captivating virtual spaces that can enhance student engagement and foster a deeper understanding of complex concepts.
The project demonstrates fundamental and advanced concepts of 3D graphics, including modeling using geometric primitives, camera positioning, and perspective projection. The scene is rendered with depth and spatial accuracy to provide an immersive visual experience.

The code for the game is written in C++, using the GLUT (OpenGL Utility Toolkit) library for windowing and input handling.

2. Computer Graphics Concepts used-

2.1. Window to Viewport Conversion:

In computer graphics, window to viewport conversion is the process of transforming a 2D or 3D scene from a world-coordinate reference frame to a screen-coordinate reference frame. In other words, it maps the scene defined in a virtual coordinate system to a real pixel-based coordinate system on the screen.

The window to viewport conversion is implemented in the reshape_callback( ) function. This function is called whenever the window is resized or reshaped. It takes the new width and height of the window as input parameters (in pixels), and sets up the projection matrix and viewport accordingly.

The projection matrix is set up using the glOrtho( ) function, which specifies an orthographic projection with left, right, bottom, top, near and far clipping planes.

The viewport is set up using the glViewport( ) function, which specifies the position and size of the viewport in pixels. In this project, the viewport is set up to cover the entire window, with the origin at the bottom-left corner of the window.

2.2. OpenGL Graphics Primitives:

Graphics primitives are basic geometric shapes or elements used to create graphics or images. These primitives are the building blocks for more complex graphics, and they include points, lines, curves, polygons, and other basic shapes. Graphics primitives can be used to create 2D or 3D graphics, and they are often used in computer graphics programming to create images, animations, and other visual effects.

This project uses several graphics primitives provided by the OpenGL library, which are used to draw the game grid, snake, and food. The primitives used in this code are:

GL_QUADS: This primitive is used to draw quadrilaterals. In this code, it is used to draw the snake and food. It takes four vertices as input and draws a filled quadrilateral.

GL_LINES: This primitive is used to draw lines. In this code, it is used to draw the grid lines. It takes two vertices as input and draws a line segment between them.

Both primitives require specifying the vertices' coordinates, which are specified using the glVertex2d( ) function. The glLineWidth( ) function is used to set the width of lines drawn by GL_LINES. The glColor3f() function is used to set the current color. It takes three arguments representing the red, green, and blue components of the color, each ranging from 0.0 to 1.0.

In addition to these primitives, the code also uses the glLoadIdentity() function, which resets the current transformation matrix to the identity matrix. This function is used to ensure that the transformations applied to previous objects do not affect the rendering of subsequent objects.

2.3. Scope of Project-

The scope of this project is to design and develop a **3D model of a smart classroom** using OpenGL in C++. It aims to simulate a modern educational environment that reflects current trends in digital and interactive learning spaces. The simulation focuses on visual realism, structural accuracy, and spatial arrangement of classroom components.

Key areas covered within the project scope include:

- Creating a **3D environment** with desks, chairs, walls, floors, and ceilings.
- Modeling of **students** and **seating arrangements**.
- Inclusion of **modern elements** like smart boards/screens, fans, and lighting.
- Implementation of **camera and perspective settings** to navigate and visualize the scene realistically.
- Laying a foundation for future extensions such as **interactivity**, **animations**, or **virtual classroom simulations**.

Out of Scope (for current version):

- Real-time interactivity (e.g., keyboard/mouse controls).
- Detailed textures or lighting effects (e.g., shadows, material properties).
- Integration with educational content or user input.
- Network-based multi-user classrooms.

2.User Defined Functions

Geometric Functions:

1. getNormal3p(GLfloat *v1, GLfloat *v2, GLfloat *v3):
   - This function takes three pointers to vertex coordinates (‘v1’, ‘v2’, ‘v3’) representing a triangle.
   - It calculates the normal vector to the plane formed by these vertices. The normal vector is perpendicular to the triangle's surface and is crucial for lighting calculations in 3D graphics.
   - It uses vector math operations (cross product) to calculate the normal vector based on the provided vertex positions.

2. cube(GLfloat r, GLfloat g, GLfloat b):
   - This function draws a colored cube.
   - It takes arguments for red (‘r’), green (‘g’), and blue (‘b’) color components, allowing for various cube colors.
   - It uses OpenGL functions like ‘glBegin(GL_QUADS)’ to define a quad (four- sided polygon) for each face of the cube.
   - It iterates through the ‘v_Cube’ data structure (containing vertex coordinates) and ‘quadIndices’ data structure (defining the order of vertices for each face) to render the cube's faces with the specified color.

3. table():
   - glPushMatrix() and glPopMatrix(): These functions are used to push and pop the current model transformation matrix onto a stack.
   - This allows the code to define transformations for each leg independently without affecting other parts of the table.
   - glTranslatef(length/2, 0, 0): This translates the model along the x-axis by length/2, effectively positioning the leg at the center of the table's width.
   - glScalef(length, height, width): This scales the model along each axis by the defined length, height, and width values, creating a leg with those dimensions.
   - glTranslatef(0, 0, 0): This additional translation might be a leftover or placeholder, as it doesn’t have any effect within the glPushMatrix-glPopMatrix block.
   - cube(0, 0, 0): This calls the cube function (assumed to be defined elsewhere) to draw a black cube (color components are 0 for all). This cube represents a single leg of the table.

4. chair():
    - Similar to ‘table()’, this function draws a chair using combinations of cubes or other basic shapes.

5. full_set():
    - This function combines the functionality of ‘table()’ and ‘chair()’ to draw a table and chair together.
    - It calls both functions and positions them appropriately to create the furniture set.

6. fan_face():
    - This function draws the central part of a fan, which will be a circle, cylinder, or a more complex shape depending on the fan design.
    - It uses OpenGL functions for drawing circles, cylinders, or polygons to create the desired fan face shape.

7. stand():
    - This function draws the stand that supports the fan, a rectangular prism shape.
    - It uses cube functions with appropriate dimensions to create the stand's base, pole, and other components.

8. leg():
    - This function draws a single blade of the fan, modeling as a rectangular prism.
    - It uses cube functions with specific dimensions and positioning to create a blade shape.

9. fan_set():
    - This function combines the functionality of ‘fan_face()’, ‘stand()’, and makes multiple calls to ‘leg()’ to draw the complete fan with blades and stand.
    - It calls the individual functions in the correct order and positions them to create the assembled fan model.

10. fan(float rot):
    - This function draws a rotating fan.
    - It calls ‘fan_set()’ to render the fan model and then use OpenGL rotation functions (‘glRotatef’) to rotate the entire fan based on the provided ‘rot’ parameter (rotation angle).

11. fan_full_set():
    - This function draws multiple fans in the scene.
    - 'It calls ‘fan(rot)’ at different positions within the scene to create the desired number and arrangement of fans.

12. almari():
    - This function draws a bookshelf, using a combination of basic shapes like cubes or cuboids.
    - It will create the bookshelf’s shelves, backboard, and other components using multiple cube functions with sizes and positions.

13. board():
    - This function draws a board, a rectangle or a plane.
    - It uses OpenGL functions for drawing rectangles or defining flat surfaces to create the board.

14. window():
    - This function draws a single window, using a combination of lines and rectangles.
    - It uses OpenGL line and rectangle drawing functions to create the window frame and separate glass sections.

15. windows():
    - This function draws multiple windows in the scene.
    - It calls ‘window()’ at different positions within the scene to create the desired number and arrangement of windows.

16. projector():
    - This function draws a projector unit, possibly using a combination of cubes or prisms.
    - It creates the projector's body, lens, and other components using functions like \`cube\` with dimensions.

17. full_set_chair_table()

    Draws multiple sets of tables and chairs in a classroom arrangement by calling full_set() at various positions.

18. projector_board()
    - Draws a white projection board (screen) for the projector.

19. AC()
    - Draws an air conditioner unit.
    - Uses scaling and translation to create a rectangular box representing the AC.

20. bulb()
    - Draws a single bulb (light source).
    - Uses two cubes: one for the bulb and one for the bulb holder, with different colors and sizes.

21. bulb_set()
    - Draws multiple bulbs at different positions in the classroom.
    - Calls bulb() several times with different translations to place bulbs at various ceiling positions.

22. door()
    - Draws a door.
    - Uses scaling and translation to create a door and its handle using cubes.

23. floorWallsCeiling(float scale)
    - Draws the floor, walls, and ceiling of the classroom.
    - Uses multiple cubes with different scales and translations to represent the floor, side walls, back wall, front wall, and ceiling.

24. light()
    - Configures and enables the OpenGL lighting for the scene.
    - Sets up three light sources (GL_LIGHT0, GL_LIGHT1, GL_LIGHT2) with different positions and properties.
    - Handles toggling of ambient, diffuse, and specular components, as well as spotlight cutoff.

25. drawHead(), drawBody(), drawArm(bool isRight), drawLegs()
    - Draws the basic parts of a student (head, body, arms, legs) using cubes.
    - Each function uses scaling, translation, and coloring to represent a body part.

26. student(float x, float y, float z)

- Draws a student at a given position using the above body part functions.
- Translates to the given position, applies scaling, and calls the body part functions.

27. students_set()
    - Draws a set of students arranged in rows.
    - Calls student() multiple times with different positions to create rows of students.

28. drawImprovedHead(int studentId), drawImprovedBody(int studentId), drawImprovedArm(bool isRight, bool isWriting), drawImprovedLegs()
    - Draws improved/detailed versions of student body parts, with more features (hair, neck, colored shirts, etc.).
    - Use more colors, more cubes, and more transformations for realism.

29 improvedStudent(float x, float y, float z, int studentId)
- Draws a more detailed student at a given position, using the improved body part functions.
- Translates and scales, applies body lean and head rotation, and calls the improved body part functions.

30. improved_students_set()
    - Draws a set of improved students in rows, with varying shirt colors.

    - Calls improvedStudent() multiple times with different positions and student IDs.

31. drawTeacher(float x, float y, float z, bool isMainTeacher)
    - Draws a teacher at a given position, with different appearance for main and walking teacher.
    - Uses cubes for body, head, arms, and legs, with different colors and poses for main and walking teacher.

32. teachers_set()
    - Draws both the main teacher and the walking teacher.
    - Calls drawTeacher() for both teachers, using global variables for their positions and animation.

33. display(void)
    - Main display function for rendering the entire scene.
    - Sets up the camera, calls all the above functions to draw the classroom, students, teachers, furniture, and lighting.
34. myKeyboardFunc(unsigned char key, int x, int y)
    - Handles keyboard input for camera and lighting controls.
    - Changes rotation, camera position, and toggles lights/lighting components based on key presses.
35. animate()
    - Handles animation for the scene (fan rotation, student/teacher movement, etc.).
    - Updates global variables for animation, such as rotation angles and positions, and triggers redisplay.
36. main(int argc, char **argv)
    - Entry point of the program. Sets up OpenGL, registers callbacks, and starts the main loop.

SOURCE CODE

```
#include <GLUT/glut.h> // Mac compatible header
#include <stdlib.h>
#include <iostream>
#include <stdio.h>
#include <math.h>
using namespace std;
// Global variables
GLfloat alpha = 0.0, theta = 0.0, axis_x = 0.0, axis_y = 0.0;
GLfloat Calpha = 360.0, C_hr_alpha = 360.0;
GLboolean fRotate = true, cRotate = true, xz_rotate = false, l_on = true;
const int width = 1920;
const int height = 1080;
GLboolean amb = false, spec = false, dif = true;
bool l_on1 = true;
bool l_on2 = true;
bool l_on3 = false;
double spt_cutoff = 40;
float rot = 0;
GLfloat eyeX = 0;
GLfloat eyeY = 10;
GLfloat eyeZ = 10;
GLfloat lookX = 0;
GLfloat lookY = 10;
GLfloat lookZ = 0;
GLfloat student_head_rotation = 0.0f;
GLfloat student_body_lean = 0.0f;
GLfloat student_hand_rotation = 0.0f;
GLfloat student_writing_motion = 0.0f;
bool head_direction = true;
bool body_direction = true;
bool writing_direction = true;
GLfloat teacher1_hand_rotation = 0.0f;
GLfloat teacher2_position_x = 0.0f;
GLfloat teacher2_position_z = -5.0f;
bool teacher1_hand_direction = true;
bool teacher2_walking_direction = true;
// Cube vertex data
static GLfloat v_Cube[8][3] = {
    {0,0,0},
    {0,0,1},
    {0,1,0},
    {0,1,1},
    {1,0,0},
    {1,0,1},
    {1,1,0},
    {1,1,1}
};
// Cube face indices
static GLubyte quadIndices[6][4] = {
    {0,2,6,4},
    {1,5,7,3},
    {0,4,5,1},
    {2,3,7,6},
    {0,1,3,2},
    {4,6,7,5}
};
// Function to calculate normal of triangle
static void getNormal3p(
    GLfloat x1, GLfloat y1, GLfloat z1,
    GLfloat x2, GLfloat y2, GLfloat z2,
    GLfloat x3, GLfloat y3, GLfloat z3)
{
    GLfloat Ux = x2 - x1;
    GLfloat Uy = y2 - y1;
    GLfloat Uz = z2 - z1;
    GLfloat Vx = x3 - x1;
    GLfloat Vy = y3 - y1;
    GLfloat Vz = z3 - z1;
    GLfloat Nx = Uy * Vz - Uz * Vy;
    GLfloat Ny = Uz * Vx - Ux * Vz;
    GLfloat Nz = Ux * Vy - Uy * Vx;
    glNormal3f(Nx, Ny, Nz);
}
void cube(GLfloat colr1, GLfloat colr2, GLfloat colr3)
{
    GLfloat cube_no[] = {0, 0, 0, 1.0f};
    GLfloat cube_amb[] = {colr1 * 0.3f, colr2 * 0.3f, colr3 * 0.3f, 1.0f};
    GLfloat cube_dif[] = {colr1, colr2, colr3, 1.0f};
    GLfloat cube_spec[] = {1.0f, 1.0f, 1.0f, 1.0f};
    GLfloat cube_sh[] = {10.0f};
    glMaterialfv(GL_FRONT, GL_AMBIENT, cube_amb);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, cube_dif);
    glMaterialfv(GL_FRONT, GL_SPECULAR, cube_spec);
    glMaterialfv(GL_FRONT, GL_SHININESS, cube_sh);
    glBegin(GL_QUADS);
    for (GLint i = 0; i < 6; i++)
    {
        getNormal3p(
            v_Cube[quadIndices[i][0]][0], v_Cube[quadIndices[i][0]][1], v_Cube[quadIndices[i][0]][2],
            v_Cube[quadIndices[i][1]][0], v_Cube[quadIndices[i][1]][1], v_Cube[quadIndices[i][1]][2],
            v_Cube[quadIndices[i][2]][0], v_Cube[quadIndices[i][2]][1], v_Cube[quadIndices[i][2]][2]
        );
        for (GLint j = 0; j < 4; j++)
        {
            glVertex3fv(&v_Cube[quadIndices[i][j]][0]);
        }
    }
    glEnd();
}
void table()
{
    float length = 0.5f;
    float height = 3.0f;
    float width = 0.5f;
    // Legs
    glPushMatrix();
    glTranslatef(length/2, 0, 0);
    glScalef(length, height, width);
    glTranslatef(0, 0, 0);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(length/2, 0, 0);
    glScalef(length, height, width);
    glTranslatef(4, 0, 0);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(length/2, 0, 0);
    glScalef(length, height, width);
    glTranslatef(0, 0, 3);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(length/2, 0, 0);
    glScalef(length, height, width);
    glTranslatef(4, 0, 3);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    // Table top
    glPushMatrix();
    glScalef(length * 6.0f, height / 6.0f, width * 5.0f);
    glTranslatef(0, 6, 0);
    cube(1.0f, 0.8f, 0.4f);
    glPopMatrix();
}
void chair()
{
    float length = 0.5f;
    float height = 2.0f;
    float width = 0.5f;
    // Legs
    glPushMatrix();
    glTranslatef(length/2, 0, 0);
    glScalef(length, height, width);
    glTranslatef(0.5f, 0, -2);
    cube(1.0f, 0.8f, 0.4f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(length/2, 0, 0);
    glScalef(length, height, width);
    glTranslatef(3.5f, 0, -2);
    cube(1.0f, 0.8f, 0.4f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(length/2, 0, 0);
    glScalef(length, height, width);
    glTranslatef(0.5f, 0, -4);
    cube(1.0f, 0.8f, 0.4f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(length/2, 0, 0);
    glScalef(length, height, width);
    glTranslatef(3.5f, 0, -4);
    cube(1.0f, 0.8f, 0.4f);
    glPopMatrix();
    // Seat
    glPushMatrix();
    glTranslatef(length/2, 0, 0);
    glScalef(length * 4.0f, height / 6.0f, width * 4.0f);
    glTranslatef(0.15f, 6, -1.2f);
    cube(1.0f, 0.7f, 0.4f);
    glPopMatrix();
    // Back support
    glPushMatrix();
    glTranslatef(length/2, 0, 0);
    glScalef(length * 4.0f, height * 1.2f, width);
    glTranslatef(0.15f, 1, -5);
    cube(1.0f, 0.7f, 0.4f);
    glPopMatrix();
}
void full_set()
{
    glPushMatrix();
    table();
    glPopMatrix();
    glPushMatrix();
    chair();
    glPopMatrix();
}
void full_set_chair_table()
{
    glPushMatrix();
    full_set();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(5, 0, 0);
    full_set();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-5, 0, 0);
    full_set();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-10, 0, 0);
    full_set();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(0, 0, -6);
    full_set();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(5, 0, -6);
    full_set();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-5, 0, -6);
    full_set();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-10, 0, -6);
    full_set();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(0, 0, -12);
    full_set();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(5, 0, -12);
    full_set();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-5, 0, -12);
    full_set();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-10, 0, -12);
    full_set();
    glPopMatrix();
    glPushMatrix();
    glScalef(1.5f, 2.0f, 1.0f);
    glTranslatef(1, 0, 6);
    table();
    glPopMatrix();
}
void fan_face()
{
    glPushMatrix();
    glScalef(2.0f, 0.5f, 1.8f);
    glTranslatef(-0.4f, 19, -0.4f);
    cube(1.0f, 1.0f, 1.0f);
    glPopMatrix();
}
void stand()
{
    glPushMatrix();
    glScalef(0.5f, 5.0f, 0.5f);
    glTranslatef(0, 2, 0);
    cube(0.392f, 0.584f, 0.929f);
    glPopMatrix();
}
void leg()
{
    glPushMatrix();
    glScalef(5.0f, 0.07f, 1.8f);
    glTranslatef(0, 140, -0.3f);
    cube(0.392f, 0.584f, 0.929f);
    glPopMatrix();
}
void fan_set()
```

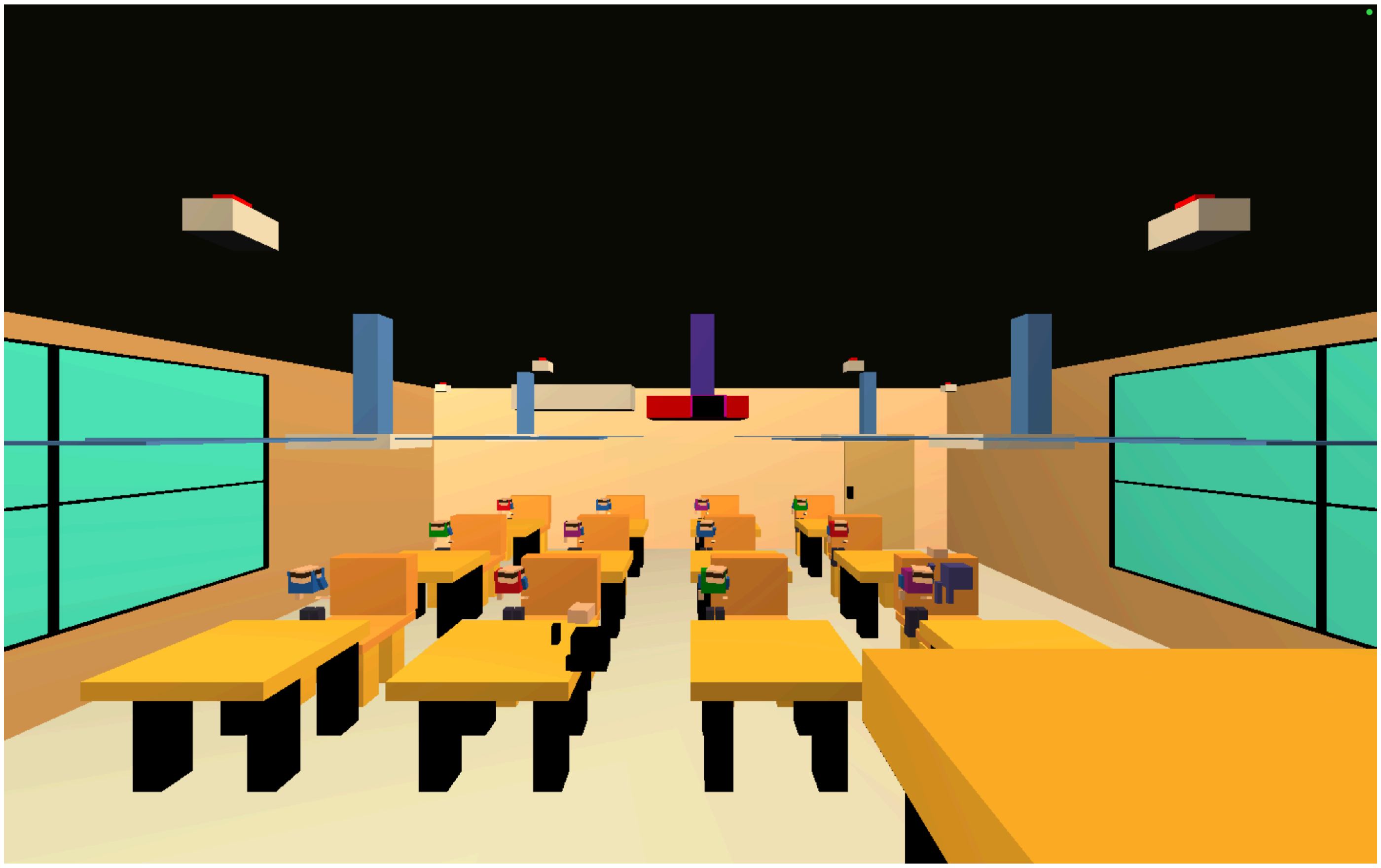
```
{
    glPushMatrix();
    fan_face();
    glPopMatrix();
    glPushMatrix();
    leg();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-5, 0.0f, 0);
    leg();
    glPopMatrix();
    glPushMatrix();
    glRotatef(90, 0, 1, 0);
    glTranslatef(-6, 0, 0);
    leg();
    glPopMatrix();
    glPushMatrix();
    glRotatef(90, 0, 1, 0);
    glTranslatef(0, 0, 0);
    leg();
    glPopMatrix();
}
void fan()
{
    glPushMatrix();
    glRotatef(alpha, 0, 0.1f, 0);
    fan_set();
    glPopMatrix();
    glPushMatrix();
    stand();
    glPopMatrix();
}
void fan_full_set()
{
    glPushMatrix();
    glTranslatef(7, 0, 0);
    fan();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-7, 0, 0);
    fan();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(7, 0, -9);
    fan();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-7, 0, -9);
    fan();
    glPopMatrix();
}
void almari()
{
    glPushMatrix();
    GLfloat almari_height = 12.0f, almari_width = 6.0f, almari_length = 3.0f;
    glPushMatrix();
    glScalef(almari_width, almari_height, almari_length);
    glTranslatef(1.0f, 0, 3.5f);
    cube(1.0f, 0.6f, 0.2f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(0.4f, 0, 4.1f);
    glScalef(almari_width / 45.0f, (almari_height - 0.5f), almari_length / 5.5f);
    glTranslatef(65.0f, 0, 11.5f);
    cube(1.0f, 0.0f, 1.0f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(1.5f, 5, 4.5f);
    glScalef(almari_width / 25.0f, almari_height / 10.0f, almari_length / 5.5f);
    glTranslatef(28.0f, 0, 10.0f);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-0.6f, 5, 4.5f);
    glScalef(almari_width / 25.0f, almari_height / 10.0f, almari_length / 5.5f);
    glTranslatef(44.0f, 0, 10.0f);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    glPopMatrix();
}
void board()
{
    glPushMatrix();
    glScalef(10.0f, 8.0f, 0.1f);
    glTranslatef(-1.2f, 0.45f, 130.0f);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
}
void window()
{
    glPushMatrix();
    glScalef(0.1f, 10.0f, 10.0f);
    glTranslatef(-140.0f, 0.3f, -0.5f);
    cube(0.5f, 1.0f, 1.0f);
    glPopMatrix();
    glPushMatrix();
    glScalef(0.1f, 10.0f, 0.1f);
    glTranslatef(-139.0f, 0.3f, 0);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    glPushMatrix();
    glScalef(0.1f, 0.1f, 10.0f);
    glTranslatef(-139.0f, 75.0f, -0.5f);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    glPushMatrix();
    glScalef(0.1f, 0.1f, 10.0f);
    glTranslatef(-139.0f, 130.0f, -0.5f);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    glPushMatrix();
    glScalef(0.1f, 0.1f, 10.0f);
    glTranslatef(-139.0f, 30.0f, -0.5f);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    glPushMatrix();
    glScalef(0.1f, 10.0f, 0.1f);
    glTranslatef(-139.0f, 0.3f, -50.0f);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    glPushMatrix();
    glScalef(0.1f, 10.0f, 0.1f);
    glTranslatef(-139.0f, 0.3f, 50.0f);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
}
void windows()
{
    glPushMatrix();
    window();
    glPopMatrix();
    glPushMatrix();
    glRotatef(180.0f, 0, 1, 0);
    window();
    glPopMatrix();
}
void projector()
{
    glPushMatrix();
    glScalef(2.0f, 0.7f, 1.8f);
    glTranslatef(-0.43f, 15.0f, -0.4f);
    cube(1.0f, 0.0f, 0.0f);
    glPopMatrix();
    glPushMatrix();
    glScalef(0.5f, 3.4f, 0.5f);
    glTranslatef(0, 3.2f, 0);
    cube(0.392f, 0.3f, 0.929f);
    glPopMatrix();
    glPushMatrix();
    glScalef(0.68f, 0.68f, 0.68f);
    glTranslatef(0, 15.45f, 1);
    cube(1.0f, 0.0f, 1.0f);
    glPopMatrix();
    glPushMatrix();
    glScalef(0.60f, 0.60f, 0.60f);
    glTranslatef(0.05f, 17.55f, 1.4f);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
}
void projector_board()
{
    glPushMatrix();
    glScalef(6.0f, 6.0f, 0.1f);
    glTranslatef(-0.2f, 0.8f, 130.0f);
    cube(1.0f, 1.0f, 1.0f);
    glPopMatrix();
}
void AC()
{
    glPushMatrix();
    glScalef(6.0f, 2.0f, 2.0f);
    glTranslatef(-1.5f, 6.0f, -7.5f);
    cube(0.8f, 0.8f, 0.8f);
    glPopMatrix();
}
void bulb()
{
    glPushMatrix();
    glScalef(0.5f, 0.5f, 0.5f);
    glTranslatef(0, 10, 0);
    cube(1.0f, 1.0f, 1.0f);
    glPopMatrix();
    glPushMatrix();
    glScalef(0.2f, 0.3f, 0.2f);
    glTranslatef(0.7f, 18.0f, 0.7f);
    cube(1.0f, 0.0f, 0.0f);
    glPopMatrix();
}
void bulb_set()
{
    glPushMatrix();
    glScalef(1.0f, 1.0f, 1.0f);
    glTranslatef(-12.0f, 8.2f, -12.0f);
    bulb();
    glPopMatrix();
    glPushMatrix();
    glScalef(1.0f, 1.0f, 1.0f);
    glTranslatef(12.0f, 8.2f, -12.0f);
    bulb();
    glPopMatrix();
    glPushMatrix();
    glScalef(1.0f, 1.0f, 1.0f);
    glTranslatef(-12.0f, 8.2f, 12.0f);
    bulb();
    glPopMatrix();
    glPushMatrix();
    glScalef(1.0f, 1.0f, 1.0f);
    glTranslatef(12.0f, 8.2f, 12.0f);
    bulb();
    glPopMatrix();
    glPushMatrix();
    glScalef(1.0f, 1.0f, 1.0f);
    glTranslatef(-5.0f, 8.2f, -5.0f);
    bulb();
    glPopMatrix();
    glPushMatrix();
    glScalef(1.0f, 1.0f, 1.0f);
    glTranslatef(5.0f, 8.2f, -5.0f);
    bulb();
    glPopMatrix();
    glPushMatrix();
    glScalef(1.0f, 1.0f, 1.0f);
    glTranslatef(-5.0f, 8.2f, 5.0f);
    bulb();
    glPopMatrix();
    glPushMatrix();
    glScalef(1.0f, 1.0f, 1.0f);
    glTranslatef(5.0f, 8.2f, 5.0f);
    bulb();
    glPopMatrix();
}
void door()
{
    glPushMatrix();
    glScalef(3.8f, 10.0f, 0.2f);
    glTranslatef(2.2f, 0, -75.0f);
    cube(0.8f, 0.7f, 0.5f);
    glPopMatrix();
    glPushMatrix();
    glScalef(0.3f, 1.0f, 0.2f);
    glTranslatef(28.0f, 4.5f, -73.0f);
    cube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
}
void floorWallsCeiling(float scale)
{
    glPushMatrix();
    glScalef(scale, 1.0f, scale);
    glTranslatef(-0.5f, -1.0f, -0.5f);
    cube(0.8f, 0.8f, 0.8f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(scale/2.0f, scale/4.0f, 0);
    glScalef(1.0f, scale/2.0f, scale);
    glTranslatef(-1.0f, -0.5f, -0.5f);
    cube(0.871f, 0.722f, 0.529f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-scale/2.0f + 1.0f, scale/4.0f, 0);
    glScalef(1.0f, scale/2.0f, scale);
    glTranslatef(-1.0f, -0.5f, -0.5f);
    cube(0.871f, 0.722f, 0.529f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(0, scale/4.0f, scale/2.0f);
    glScalef(scale, scale/2.0f, 1.0f);
    glTranslatef(-0.5f, -0.5f, -1.0f);
    cube(1.0f, 0.855f, 0.725f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(0, scale/4.0f, scale/2.0f);
    glScalef(scale, scale/2.0f, 1.0f);
    glTranslatef(-0.5f, -0.5f, -31.0f);
    cube(1.0f, 0.855f, 0.725f);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(0, scale/2.0f, 0);
    glScalef(scale, 1.0f, scale);
    glTranslatef(-0.5f, -1.0f, -0.5f);
    cube(0.871f, 0.722f, 0.529f);
    glPopMatrix();
}
void light()
{
    GLfloat l_amb[] = {0.2f, 0.2f, 0.2f, 1.0f};
    GLfloat l_dif[] = {0.961f, 0.871f, 0.702f};
    GLfloat l_spec[] = {0.2f, 0.2f, 0.2f, 1.0f};
    GLfloat l_no[] = {0.0f, 0.0f, 0.0f, 1.0f};
    GLfloat l_pos1[] = {-20.0f, 20.0f, 20.0f, 1.0f};
    GLfloat l_pos2[] = {44.0f, 30.0f, -5.0f, 1.0f};
    GLfloat l_pos3[] = {0.0f, 60.0f, 0.0f, 1.0f};
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHT1);
    glEnable(GL_LIGHT2);
    if (l_on1)
    {
        glLightfv(GL_LIGHT0, GL_AMBIENT, amb ? l_amb : l_no);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, dif ? l_dif : l_no);
        glLightfv(GL_LIGHT0, GL_SPECULAR, spec ? l_spec : l_no);
    }
    else
    {
        glLightfv(GL_LIGHT0, GL_AMBIENT, l_no);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, l_no);
        glLightfv(GL_LIGHT0, GL_SPECULAR, l_no);
    }
    glLightfv(GL_LIGHT0, GL_POSITION, l_pos1);
    if (l_on2)
    {
        glLightfv(GL_LIGHT1, GL_AMBIENT, amb ? l_amb : l_no);
        glLightfv(GL_LIGHT1, GL_DIFFUSE, dif ? l_dif : l_no);
        glLightfv(GL_LIGHT1, GL_SPECULAR, spec ? l_spec : l_no);
    }
    else
    {
        glLightfv(GL_LIGHT1, GL_AMBIENT, l_no);
        glLightfv(GL_LIGHT1, GL_DIFFUSE, l_no);
        glLightfv(GL_LIGHT1, GL_SPECULAR, l_no);
    }
    glLightfv(GL_LIGHT1, GL_POSITION, l_pos2);
    if (l_on3)
    {
        glLightfv(GL_LIGHT2, GL_AMBIENT, amb ? l_amb : l_no);
        glLightfv(GL_LIGHT2, GL_DIFFUSE, dif ? l_dif : l_no);
        glLightfv(GL_LIGHT2, GL_SPECULAR, spec ? l_spec : l_no);
    }
    else
    {
        glLightfv(GL_LIGHT2, GL_AMBIENT, l_no);
        glLightfv(GL_LIGHT2, GL_DIFFUSE, l_no);
        glLightfv(GL_LIGHT2, GL_SPECULAR, l_no);
    }
    glLightfv(GL_LIGHT2, GL_POSITION, l_pos3);
    GLfloat spot_direction[] = {0.0f, -1.0f, 0.0f};
    glLightfv(GL_LIGHT2, GL_SPOT_DIRECTION, spot_direction);
    GLfloat spt_ct[] = {static_cast<GLfloat>(spt_cutoff)};
    glLightfv(GL_LIGHT2, GL_SPOT_CUTOFF, spt_ct);
}
void drawHead() {
    glPushMatrix();
    glScalef(0.8f, 0.8f, 0.8f);
    glTranslatef(0.0f, 4.5f, 0.0f);
    cube(0.96f, 0.80f, 0.69f);  // Skin color
    glPopMatrix();
}
void drawBody() {
    glPushMatrix();
    glScalef(1.0f, 1.5f, 0.5f);
    glTranslatef(-0.4f, 2.0f, 0.0f);
    cube(0.2f, 0.2f, 0.8f);  // Blue shirt
    glPopMatrix();
}
void drawArm(bool isRight) {
    glPushMatrix();
    if(isRight) {
        glTranslatef(0.6f, 3.0f, 0.0f);
        glRotatef(student_hand_rotation, 1, 0, 0);
    } else {
        glTranslatef(-0.6f, 3.0f, 0.0f);
        glRotatef(-student_hand_rotation, 1, 0, 0);
    }
    glScalef(0.3f, 1.0f, 0.3f);
    cube(0.96f, 0.80f, 0.69f);  // Skin color
    glPopMatrix();
}
void drawLegs() {
    // Left leg
    glPushMatrix();
    glScalef(0.4f, 1.5f, 0.4f);
    glTranslatef(-0.5f, 0.0f, 0.0f);
    cube(0.2f, 0.2f, 0.2f);  // Dark pants
    glPopMatrix();
    // Right leg
    glPushMatrix();
    glScalef(0.4f, 1.5f, 0.4f);
    glTranslatef(0.5f, 0.0f, 0.0f);
    cube(0.2f, 0.2f, 0.2f);  // Dark pants
    glPopMatrix();
}
void student(float x, float y, float z) {
    glPushMatrix();
    glTranslatef(x, y, z);
    glScalef(0.5f, 0.5f, 0.5f);

    // Head with rotation
    glPushMatrix();
    glRotatef(student_head_rotation, 0, 1, 0);
    drawHead();
    glPopMatrix();

    drawBody();
    drawArm(true);   // Right arm
    drawArm(false);  // Left arm
    drawLegs();
    glPopMatrix();
}
void students_set() {
    // First row
    student(-9.5f, 2.0f, -1.5f);
    student(-4.5f, 2.0f, -1.5f);
    student(0.5f, 2.0f, -1.5f);
    student(5.5f, 2.0f, -1.5f);
    // Second row
    student(-9.5f, 2.0f, -7.5f);
    student(-4.5f, 2.0f, -7.5f);
    student(0.5f, 2.0f, -7.5f);
    student(5.5f, 2.0f, -7.5f);
    // Third row
    student(-9.5f, 2.0f, -13.5f);
    student(-4.5f, 2.0f, -13.5f);
    student(0.5f, 2.0f, -13.5f);
    student(5.5f, 2.0f, -13.5f);
}
// Colors for student features
const GLfloat SKIN_COLOR[] = {0.96f, 0.80f, 0.69f, 1.0f};
const GLfloat HAIR_COLOR[] = {0.2f, 0.15f, 0.1f, 1.0f};
const GLfloat SHIRT_COLORS[][4] = {
    {0.2f, 0.4f, 0.8f, 1.0f},  // Blue
    {0.8f, 0.2f, 0.2f, 1.0f},  // Red
    {0.2f, 0.6f, 0.2f, 1.0f},  // Green
    {0.6f, 0.2f, 0.6f, 1.0f}   // Purple
};
const GLfloat PANTS_COLOR[] = {0.2f, 0.2f, 0.3f, 1.0f};
void drawImprovedHead(int studentId) {
    glPushMatrix();

    // Head
    glPushMatrix();
    glScalef(0.7f, 0.8f, 0.7f);
    glTranslatef(0.0f, 4.8f, 0.0f);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, SKIN_COLOR);
    cube(SKIN_COLOR[0], SKIN_COLOR[1], SKIN_COLOR[2]);
    glPopMatrix();
    // Hair
    glPushMatrix();
    glScalef(0.72f, 0.3f, 0.72f);
    glTranslatef(0.0f, 14.0f, 0.0f);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, HAIR_COLOR);
    cube(HAIR_COLOR[0], HAIR_COLOR[1], HAIR_COLOR[2]);
    glPopMatrix();
    // Neck
    glPushMatrix();
    glScalef(0.3f, 0.3f, 0.3f);
    glTranslatef(0.0f, 13.0f, 0.0f);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, SKIN_COLOR);
    cube(SKIN_COLOR[0], SKIN_COLOR[1], SKIN_COLOR[2]);
    glPopMatrix();
    glPopMatrix();
}
void drawImprovedBody(int studentId) {
    glPushMatrix();

    // Torso
    glPushMatrix();
    glScalef(1.0f, 1.4f, 0.6f);
    glTranslatef(-0.4f, 2.2f, 0.0f);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, SHIRT_COLORS[studentId % 4]);
    cube(SHIRT_COLORS[studentId % 4][0], SHIRT_COLORS[studentId % 4][1], SHIRT_COLORS[studentId % 4][2]);
    glPopMatrix();
    glPopMatrix();
}
void drawImprovedArm(bool isRight, bool isWriting) {
    glPushMatrix();

    float armRotation = isWriting ? (isRight ? student_writing_motion : 0) : student_hand_rotation;

    if(isRight) {
        glTranslatef(0.5f, 3.2f, 0.0f);
        glRotatef(armRotation, 1, 0, 0);
    } else {
        glTranslatef(-0.5f, 3.2f, 0.0f);
        glRotatef(-armRotation/2, 1, 0, 0);
    }
    // Upper arm
    glPushMatrix();
    glScalef(0.25f, 0.8f, 0.25f);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, SHIRT_COLORS[0]);
    cube(SHIRT_COLORS[0][0], SHIRT_COLORS[0][1], SHIRT_COLORS[0][2]);
    glPopMatrix();
    // Lower arm
    glPushMatrix();
    glTranslatef(0.0f, -0.8f, 0.0f);
    glRotatef(isWriting && isRight ? 45 : 0, 1, 0, 0);
    glScalef(0.2f, 0.6f, 0.2f);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, SKIN_COLOR);
    cube(SKIN_COLOR[0], SKIN_COLOR[1], SKIN_COLOR[2]);
    glPopMatrix();
    glPopMatrix();
}
void drawImprovedLegs() {
    // Left leg
    glPushMatrix();
    glScalef(0.35f, 1.8f, 0.35f);
    glTranslatef(-0.6f, 0.0f, 0.0f);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, PANTS_COLOR);
    cube(PANTS_COLOR[0], PANTS_COLOR[1], PANTS_COLOR[2]);
    glPopMatrix();
    // Right leg
    glPushMatrix();
    glScalef(0.35f, 1.8f, 0.35f);
    glTranslatef(0.6f, 0.0f, 0.0f);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, PANTS_COLOR);
    cube(PANTS_COLOR[0], PANTS_COLOR[1], PANTS_COLOR[2]);
    glPopMatrix();
}
void improvedStudent(float x, float y, float z, int studentId) {
    glPushMatrix();
    glTranslatef(x, y, z);
    glScalef(0.6f, 0.6f, 0.6f);

    // Apply body lean
    glRotatef(student_body_lean, 1, 0, 0);

    // Head with rotation
    glPushMatrix();
    glRotatef(student_head_rotation, 0, 1, 0);
    drawImprovedHead(studentId);
    glPopMatrix();

    drawImprovedBody(studentId);
    drawImprovedArm(true, true);    // Right arm (writing)
    drawImprovedArm(false, false);  // Left arm
    drawImprovedLegs();

    glPopMatrix();
}
void improved_students_set() {
    // First row
    improvedStudent(-9.5f, 2.0f, -1.5f, 0);
    improvedStudent(-4.5f, 2.0f, -1.5f, 1);
    improvedStudent(0.5f, 2.0f, -1.5f, 2);
    improvedStudent(5.5f, 2.0f, -1.5f, 3);
    // Second row
    improvedStudent(-9.5f, 2.0f, -7.5f, 2);
    improvedStudent(-4.5f, 2.0f, -7.5f, 3);
    improvedStudent(0.5f, 2.0f, -7.5f, 0);
    improvedStudent(5.5f, 2.0f, -7.5f, 1);
    // Third row
    improvedStudent(-9.5f, 2.0f, -13.5f, 1);
    improvedStudent(-4.5f, 2.0f, -13.5f, 0);
    improvedStudent(0.5f, 2.0f, -13.5f, 3);
    improvedStudent(5.5f, 2.0f, -13.5f, 2);
}
const GLfloat TEACHER1_COLOR[] = {0.0f, 0.0f, 0.0f, 1.0f};  // Black suit
const GLfloat TEACHER2_COLOR[] = {0.3f, 0.3f, 0.6f, 1.0f};  // Navy blue suit
void drawTeacher(float x, float y, float z, bool isMainTeacher) {
    glPushMatrix();
    glTranslatef(x, y, z);
    glScalef(0.5f, 0.5f, 0.5f);  // Teachers slightly larger than students
```

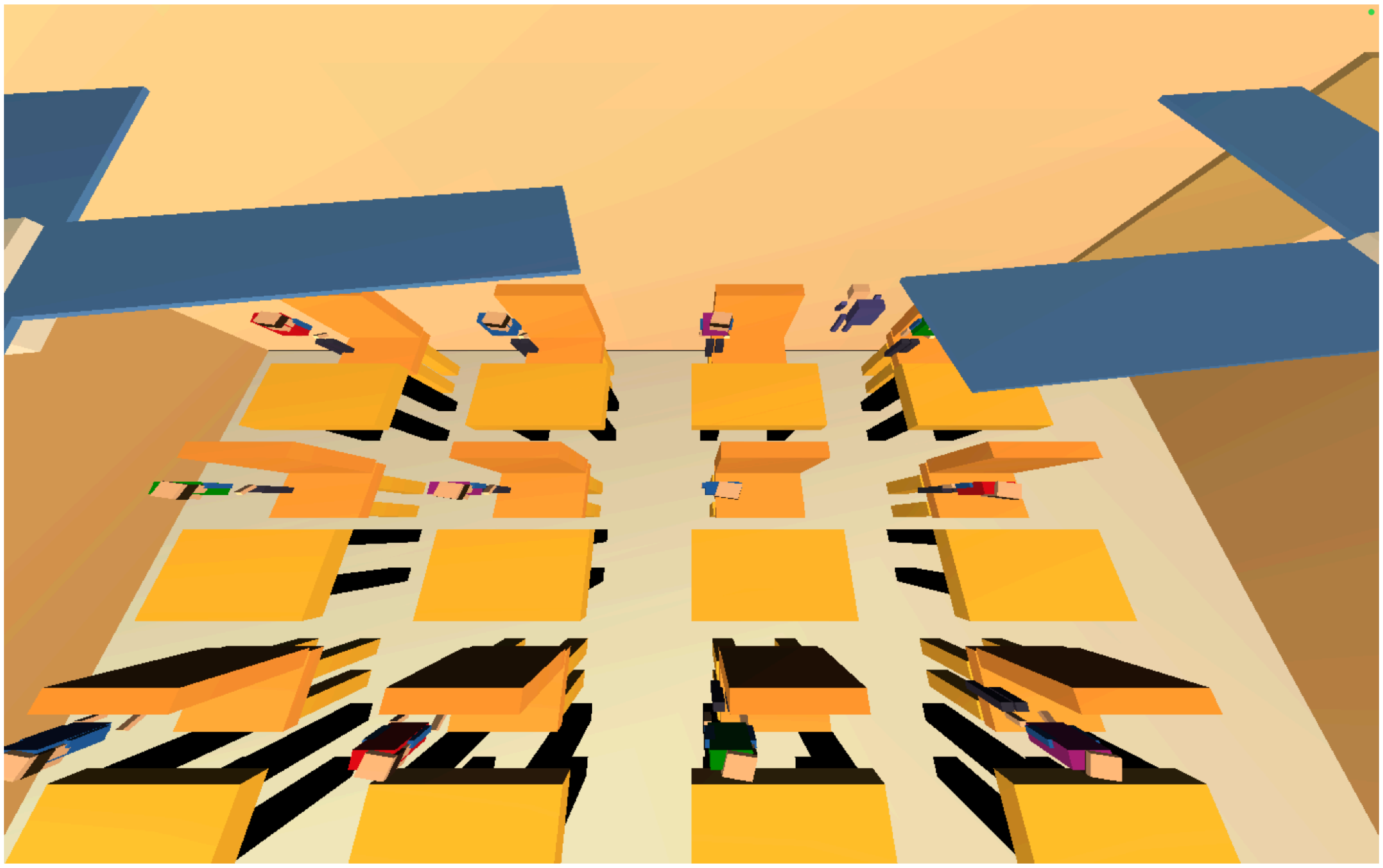
```
// Body - taller and more formal looking
glPushMatrix();
glTranslatef(0, 3, 0);
glScalef(1.2f, 1.8f, 0.8f);  // Taller body
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, isMainTeacher ? TEACHER1_COLOR : TEACHER2_COLOR);
cube(isMainTeacher ? TEACHER1_COLOR[0] : TEACHER2_COLOR[0],
    isMainTeacher ? TEACHER1_COLOR[1] : TEACHER2_COLOR[1],
    isMainTeacher ? TEACHER1_COLOR[2] : TEACHER2_COLOR[2]);
glPopMatrix();

// Head
glPushMatrix();
glTranslatef(0, 5.5f, 0);
if(isMainTeacher) {
   glRotatef(student_head_rotation/2, 0, 1, 0);  // Slower head movement for main teacher
} else {
   glRotatef(teacher2_walking_direction ? 90 : -90, 0, 1, 0);  // Walking teacher looks left/right
}
glScalef(0.7f, 0.7f, 0.7f);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, SKIN_COLOR);
cube(SKIN_COLOR[0], SKIN_COLOR[1], SKIN_COLOR[2]);
glPopMatrix();

// Right arm
glPushMatrix();
glTranslatef(0.8f, 4.0f, 0.0f);
if(isMainTeacher) {
   glRotatef(teacher1_hand_rotation, 1, 0, 0);  // Teaching gesture
} else {
   glRotatef(-20, 1, 0, 0);  // Walking position
}
glScalef(0.25f, 1.0f, 0.25f);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, isMainTeacher ? TEACHER1_COLOR : TEACHER2_COLOR);
cube(isMainTeacher ? TEACHER1_COLOR[0] : TEACHER2_COLOR[0],
    isMainTeacher ? TEACHER1_COLOR[1] : TEACHER2_COLOR[1],
    isMainTeacher ? TEACHER1_COLOR[2] : TEACHER2_COLOR[2]);
glPopMatrix();

// Left arm
glPushMatrix();
glTranslatef(-0.8f, 4.0f, 0.0f);
if(isMainTeacher) {
   glRotatef(-teacher1_hand_rotation/2, 1, 0, 0);  // Subtle movement
} else {
   glRotatef(20, 1, 0, 0);  // Walking position
}
glScalef(0.25f, 1.0f, 0.25f);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, isMainTeacher ? TEACHER1_COLOR : TEACHER2_COLOR);
cube(isMainTeacher ? TEACHER1_COLOR[0] : TEACHER2_COLOR[0],
    isMainTeacher ? TEACHER1_COLOR[1] : TEACHER2_COLOR[1],
    isMainTeacher ? TEACHER1_COLOR[2] : TEACHER2_COLOR[2]);
glPopMatrix();

// Legs
glPushMatrix();
```

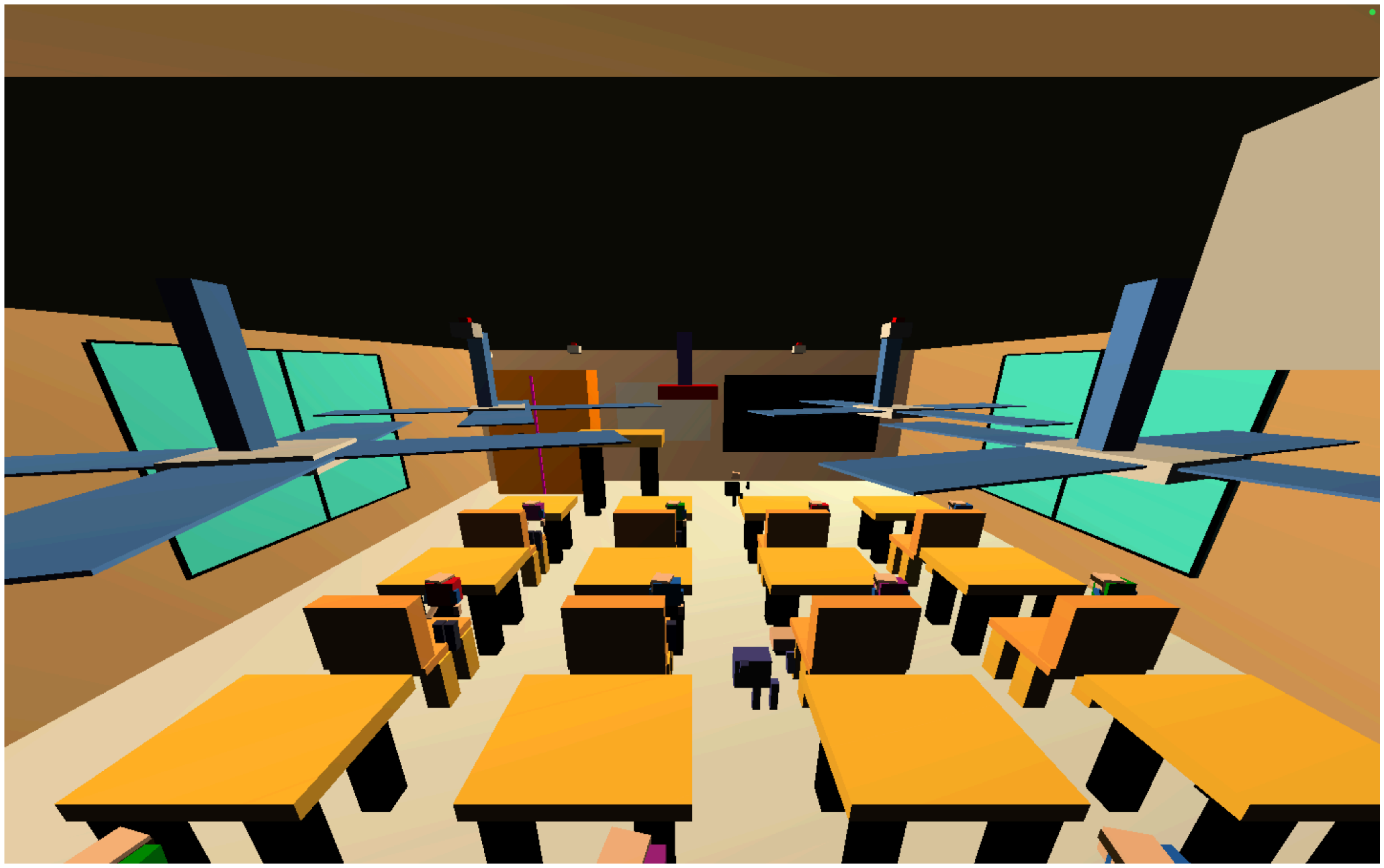
```
    glTranslatef(0.3f, 1.5f, 0);
    glScalef(0.25f, 1.8f, 0.3f);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, isMainTeacher ? TEACHER1_COLOR : TEACHER2_COLOR);
    cube(isMainTeacher ? TEACHER1_COLOR[0] : TEACHER2_COLOR[0],
        isMainTeacher ? TEACHER1_COLOR[1] : TEACHER2_COLOR[1],
        isMainTeacher ? TEACHER1_COLOR[2] : TEACHER2_COLOR[2]);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-0.3f, 1.5f, 0);
    glScalef(0.25f, 1.8f, 0.3f);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, isMainTeacher ? TEACHER1_COLOR : TEACHER2_COLOR);
    cube(isMainTeacher ? TEACHER1_COLOR[0] : TEACHER2_COLOR[0],
        isMainTeacher ? TEACHER1_COLOR[1] : TEACHER2_COLOR[1],
        isMainTeacher ? TEACHER1_COLOR[2] : TEACHER2_COLOR[2]);
    glPopMatrix();

    glPopMatrix();
}
void teachers_set() {
    // Main teacher near the board
    drawTeacher(-2.0f, 2.2f, 2.0f, true);

    // Walking teacher (position updated in animate)
    drawTeacher(teacher2_position_x, 2.2f, teacher2_position_z, false);
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-3.0f, 3.0f, -3.0f, 3.0f, 2.0f, 100.0f);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(eyeX, eyeY, eyeZ, lookX, lookY, lookZ, 0, 1, 0);
    glRotatef(rot, 0, 1, 0);
    light();
    fan_full_set();
    full_set_chair_table();
    floorWallsCeiling(30.0f);
    windows();
    board();
    almari();
    projector();
    AC();
    projector_board();
    bulb_set();
    door();
    improved_students_set();
    teachers_set();
    glFlush();
    glutSwapBuffers();
}
void myKeyboardFunc(unsigned char key, int x, int y)
{
   switch (key)
    {
    case 'd':
        rot++;
        break;
    case 'a':
        rot--;
        break;
    case 'w':
        eyeY++;
        break;
    case 's':
        eyeY--;
        break;
    case 'z':
        eyeX = 0;
        eyeZ++;
        lookZ++;
        break;
    case 'x':
        eyeZ--;
        break;
    case 'r':
        l_on1 = true;
        l_on2 = true;
        l_on3 = false;
        spt_cutoff = 40;
        rot = 0;
        eyeX = 0;
        eyeY = 10;
        eyeZ = 10;
        lookX = 0;
        lookY = 10;
        lookZ = 0;
        break; // ✅ Break was missing after 'r'!
    case '1':
        l_on1 = !l_on1;
        break;
    case '2':
        l_on2 = !l_on2;
        break;
    case '3':
        l_on3 = !l_on3;
        break;
    case '4':
        amb = !amb;
        break;
    case '5':
        spec = !spec;
        break;
    case '6':
        dif = !dif;
        break;
    }
}
```

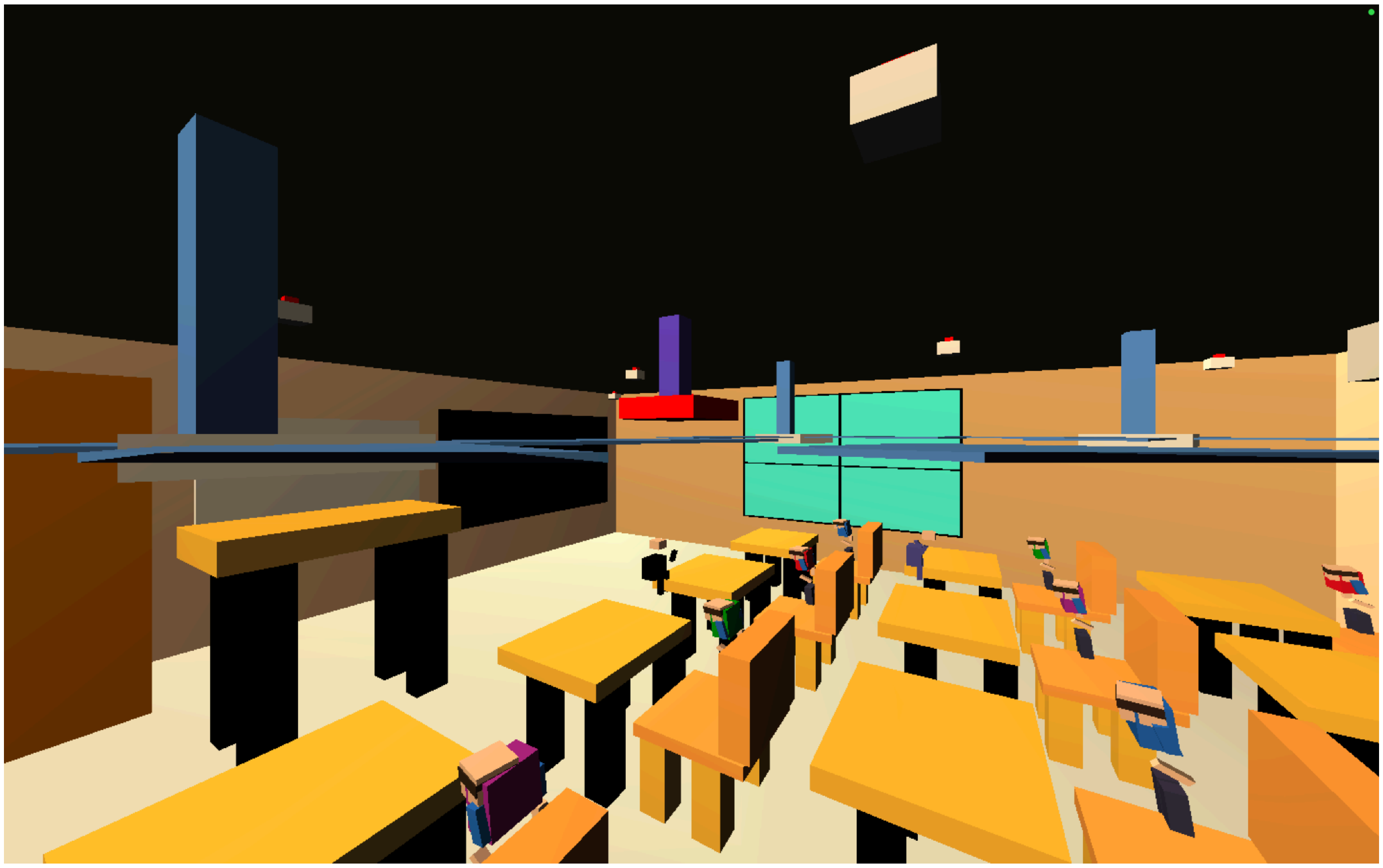
```
// Update the animate function
void animate() {
    if (fRotate == true) {
        alpha += 5;
        if (alpha > 360.0f)
            alpha -= 360.0f * floor(alpha / 360.0f);
    }
    // Student head animation (looking around)
    if (head_direction) {
        student_head_rotation += 0.3f;
        if (student_head_rotation >= 20.0f)
            head_direction = false;
    } else {
        student_head_rotation -= 0.3f;
        if (student_head_rotation <= -20.0f)
            head_direction = true;
    }
    // Student body lean animation
    if (body_direction) {
        student_body_lean += 0.1f;
        if (student_body_lean >= 5.0f)
            body_direction = false;
    } else {
        student_body_lean -= 0.1f;
        if (student_body_lean <= -2.0f)
            body_direction = true;
    }
    // Writing animation
    if (writing_direction) {
        student_writing_motion += 0.5f;
        if (student_writing_motion >= 15.0f)
            writing_direction = false;
    } else {
        student_writing_motion -= 0.5f;
        if (student_writing_motion <= -5.0f)
            writing_direction = true;
    }
    if (teacher1_hand_direction) {
            teacher1_hand_rotation += 0.4f;
            if (teacher1_hand_rotation >= 30.0f)
                teacher1_hand_direction = false;
        } else {
            teacher1_hand_rotation -= 0.4f;
            if (teacher1_hand_rotation <= -10.0f)
                teacher1_hand_direction = true;
        }
        // Walking teacher movement
        if (teacher2_walking_direction) {
            teacher2_position_x += 0.05f;
            if (teacher2_position_x >= 8.0f) {
                teacher2_walking_direction = false;
                teacher2_position_z -= 2.0f;  // Move to next row
                if (teacher2_position_z < -14.0f) {
                    teacher2_position_z = -2.0f;  // Reset to front
                }
            }
        } else {
            teacher2_position_x -= 0.05f;
            if (teacher2_position_x <= -8.0f) {
                teacher2_walking_direction = true;
                teacher2_position_z -= 2.0f;  // Move to next row
                if (teacher2_position_z < -14.0f) {
                    teacher2_position_z = -2.0f;  // Reset to front
                }
            }
        }
    glutPostRedisplay();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(width, height);
    glutCreateWindow("Class Room Render 3D");
    glShadeModel(GL_SMOOTH);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_NORMALIZE);
    glEnable(GL_LIGHTING);
    glutKeyboardFunc(myKeyboardFunc);
    glutDisplayFunc(display);
    glutIdleFunc(animate);
    glutMainLoop();
    return 0; // ✅ Mac expects return from main!
}
```

PROJECT SCREENSHOTS









References

1. OpenGl Concepts from Geeksforgeeks
   https://www.geeksforgeeks.org/getting-started-with-opengl/

2. Pearson Book- computer graphics with openGl (Hearn and Baker)
   https://books.google.co.in/books?hl=en&lr=&id=4D9IqeflmswC&oi=fnd&pg=PA2&dq=opengl&ots=9VLFkwKftN&sig=GCmvD_iwwz2gGrf15ItFQRE74v0&redir_esc=y#v=onepage&q=opengl&f=false

3. OpenGl Window to Viewport Coordinate Transformation
   https://www.tutorialspoint.com/computer_graphics/window_viewport_coordinate_transformation.htm
4. OpenGL Drawing Primitives
   https://www.dgp.toronto.edu/~ah/csc418/fall_2001/tut/ogl_draw.html#:~:text=OpenGL%20supports%20several%20basic%20primitive,using%20a%20sequence%20of%20vertices.