Computer Graphics Practicals

Mini Project

Campus 3D Basic Simulation

U18CO081
Krunal Rank

Description:

This project involves a Simulation of Basic Campus 3D Space. It has buildings, gates, a temple, a football stadium, a basketball stadium and a badminton stadium. Basic necessities of the project such as using 3D Objects, Transformations, Clipping, User Interaction and Shading have been implemented using available OpenGL Functions.

It has been implemented in C++ using some libraries. Necessary comments have been added in the code wherever required.

Some libraries that have been used are:

- OpenGL (GL, GLUT, GLU)
- Standard IO
- STL

Code:

```
#include <stdio.h>
#include <math.h>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <errno.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <bits/stdc++.h>
using namespace std;
// To Compile:-
// g++ -pthread 1.cpp -lglfw3 -lGLEW -lGLU -lGL -lXrandr -lXxf86vm -lXi -lXinerama
-lX11 -lrt -ldl -lglut

float y = 1, x = -30, z = 100; // Initial Coordinates of Camera
float deltaMove = 0.0,deltaMoveCross = 0.0;          // Camera Moves
// Camera direction
float lx = 0.0, lz = -1; // Camera Orientation
float angle = 0.0;       // Camera Rotation
float deltaAngle = 0.0; // Camera Drag Difference

// Mouse Drag Variables
int isDragging = 0;
int xDragStart = 0;
int move = 0;
int vertmove = 0;
int toggle = 0;

unsigned char header[54]; // Each BMP file begins by a 54-bytes header
unsigned int dataPos;     // Position in the file where the actual data begins
unsigned int width, height;
unsigned int imageSize;   // = width*height*3
unsigned char *data = NULL; // Actual RGB data

void draw_board()
{

   glColor3f(0.177, 0.564, 1);

   // Board Legs
   glBegin(GL_QUADS);
   glVertex3f(0, 0, 0);
   glVertex3f(1, 0, 0);
   glVertex3f(1, 2, 0);
   glVertex3f(0, 2, 0);
   glVertex3f(9, 0, 0);
   glVertex3f(10, 0, 0);
   glVertex3f(10, 2, 0);
   glVertex3f(9, 2, 0);
   glEnd();

   // Board
   glColor3f(0.690, 0.878, 0.901);
   glBegin(GL_QUADS);
   glVertex3f(0, 2, 0);
   glVertex3f(10, 2, 0);
   glVertex3f(10, 4, 0);
   glVertex3f(0, 4, 0);
   glEnd();
}

GLuint loadBMP_custom(const char *imagepath)
{
   // Open the file
   FILE *file = fopen(imagepath, "rb");
   if (!file)
   {
       printf("Image could not be opened\n");
```

```
        printf("Error %d \n", errno);
        return 0;
    }
    if (fread(header, 1, 54, file) != 54)
    { // If not 54 bytes read : problem
        printf("Not a correct BMP file\n");
        return false;
    }
    if (header[0] != 'B' || header[1] != 'M')
    {
        printf("Not a correct BMP file\n");
        return 0;
    }
    dataPos = *(int *)&(header[0x0A]);
    imageSize = *(int *)&(header[0x22]);
    width = *(int *)&(header[0x12]);
    height = *(int *)&(header[0x16]);
    // Some BMP files are misformatted, guess missing information
    if (imageSize == 0)
        imageSize = width * height * 3; // 3 : one byte for each Red, Green and Blue
component
    if (dataPos == 0)
        dataPos = 54; // The BMP header is done that way
    // Create a buffer
    data = new unsigned char[imageSize];
    // Read the actual data from the file into the buffer
    fread(data, 1, imageSize, file);
    //Everything is in memory now, the file can be closed
    fclose(file);
}

void draw_map()
{
    GLuint Texture = loadBMP_custom("field.bmp");
    glEnable(GL_TEXTURE_2D);
    GLuint textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);
    // Set the texture wrapping parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    // Set texture filtering parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    // Load image, create texture and generate mipmaps
```

```
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE,
data);

   glColor3f(1.0f, 1.0f, 1.0f);
   glBegin(GL_QUADS);
   glTexCoord2f(0.0f, 0.0f);
   glVertex3f(-15.0f, -3.0f, 20.0f);
   glTexCoord2f(10.0f, 0.0f);
   glVertex3f(30.0f, -3.0f, 20.0f);
   glTexCoord2f(10.0f, 10.0f);
   glVertex3f(30.0f, -3.0f, -20.0f);
   glTexCoord2f(0.0f, 10.0f);
   glVertex3f(-10.0f, -3.0f, -20.0f);
   glEnd();

   glBindTexture(GL_TEXTURE_2D, 0); // Unbind texture when done, so we won't
accidently mess up our texture.
   delete[] data;
   glDeleteTextures(1, &textureID);
   glDisable(GL_TEXTURE_2D);
}

void draw_idol()
{
   GLuint Texture = loadBMP_custom("Vishnu.bmp");
   glEnable(GL_TEXTURE_2D);
   GLuint textureID;
   glGenTextures(1, &textureID);
   glBindTexture(GL_TEXTURE_2D, textureID);
   // Set the texture wrapping parameters
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   // Set texture filtering parameters
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
   // Load image, create texture and generate mipmaps
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 1, GL_RGB, GL_UNSIGNED_BYTE,
data);

   glColor3f(1.0f, 1.0f, 1.0f);
   glBegin(GL_QUADS);
   glTexCoord2f(0.0f, 0.0f);
   glVertex3f(0.0f, .0f, 0.0f);
   glTexCoord2f(1.0f, 0.0f);
   glVertex3f(4.0f, 0.0f, 0.0f);
```

```
    glTexCoord2f(1.0f, 1.0f);
    glVertex3f(4.0f, 5.0f, 0.0f);
    glTexCoord2f(0.0f, 1.0f);
    glVertex3f(0.0f, 5.0f, 0.0f);
    glEnd();

    glBindTexture(GL_TEXTURE_2D, 0); // Unbind texture when done, so we won't
accidentily mess up our texture.
    delete[] data;
    glDeleteTextures(1, &textureID);
    glDisable(GL_TEXTURE_2D);
}

// World Collision
void restrict()
{
    if (x > 100)
        x = 100;
    else if (x < -100)
        x = -100;
    if (y > 60)
        y = 60;
    else if (y < 0.5)
        y = 0.5;
    if (z > 100)
        z = 100;
    else if (z < -100)
        z = -100;
}

// Badmintion Court
void mech_court()
{
    int k;
    glPushMatrix();
    glTranslatef(-70, 0.1, 20);
    glColor3f(0.5, 0.5, 0.5);
    glBegin(GL_QUADS);
    glVertex3f(0, 0, 0);
    glVertex3f(20, 0, 0);
    glVertex3f(20, 0, -40);
    glVertex3f(0, 0, -40);
    glEnd();
    glColor3f(0, 0, 0);
    for (k = 2; k < 18; k += 4)
    {
        glBegin(GL_QUADS);
        glVertex3f(k, 0, -0.1);
        glVertex3f(k, 4, -0.1);
        glVertex3f(k + 2, 4, -0.1);
        glVertex3f(k + 2, 0, -0.1);
        glEnd();
    }
    for (k = -2; k > -38; k -= 6)
    {
        glBegin(GL_QUADS);
        glVertex3f(0.1, 0, k);
        glVertex3f(0.1, 4, k);
        glVertex3f(0.1, 4, k - 2);
        glVertex3f(0.1, 0, k - 2);
        glEnd();
    }
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-50, 0.1, 20);
    for (k = -2; k > -38; k -= 6)
    {
        glBegin(GL_QUADS);
        glVertex3f(-0.1, 0, k);
        glVertex3f(-0.1, 4, k);
        glVertex3f(-0.1, 4, k - 2);
        glVertex3f(-0.1, 0, k - 2);
        glEnd();
    }
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-50, 0.1, -20);
    for (k = -2; k > -18; k -= 4)
    {
        glBegin(GL_QUADS);
        glVertex3f(k, 0, 0.1);
        glVertex3f(k, 4, 0.1);
        glVertex3f(k - 2, 4, 0.1);
        glVertex3f(k - 2, 0, 0.1);
        glEnd();
    }
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-65, 0.1, 15);
    glColor3f(0, 0, 1);
```

```
glBegin(GL_QUADS);
glVertex3f(0, 0.2, 0);
glVertex3f(10, 0.2, 0);
glVertex3f(10, 0.2, -30);
glVertex3f(0, 0.2, -30);
glEnd();
glColor3f(1, 1, 1);
glBegin(GL_LINE_LOOP);
glVertex3f(1, 0.3, -1);
glVertex3f(9, 0.3, -1);
glVertex3f(9, 0.3, -13);
glVertex3f(1, 0.3, -13);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex3f(1, 0.3, -15);
glVertex3f(9, 0.3, -15);
glVertex3f(9, 0.3, -29);
glVertex3f(1, 0.3, -29);
glEnd();
glPopMatrix();
glPushMatrix();
glTranslatef(-65, 0, 8);
glColor3f(1, 1, 1);
for (float i = 0; i < 0.8; i += 0.2)
    for (float j = 0; j < 9.8; j += 0.2)
    {
        glBegin(GL_LINE_LOOP);
        glVertex3f(j, i, 0);
        glVertex3f(j + 0.2, i, 0);
        glVertex3f(j + 0.2, i + 0.2, 0);
        glVertex3f(j, i + 0.2, 0);
        glEnd();
    }
glPopMatrix();
glPushMatrix();
glTranslatef(-65, 0, -8);
glColor3f(1, 1, 1);
for (float i = 0; i < 0.8; i += 0.2)
    for (float j = 0; j < 9.8; j += 0.2)
    {
        glBegin(GL_LINE_LOOP);
        glVertex3f(j, i, 0);
        glVertex3f(j + 0.2, i, 0);
        glVertex3f(j + 0.2, i + 0.2, 0);
        glVertex3f(j, i + 0.2, 0);
```

```
            glEnd();
        }
    glPopMatrix();
    for (int i = 0; i < 15; i += 5)
    {
        glPushMatrix();
        glTranslatef(-70, 6 + i, 20);
        glColor3f(1, 0.894, 0.709);
        glBegin(GL_QUADS);
        glVertex3f(0, 0, 0);
        glVertex3f(20, 0, 0);
        glVertex3f(20, 0, -4);
        glVertex3f(0, 0, -4);
        glEnd();
        glColor3f(1, 0.972, 0.862);
        glBegin(GL_QUADS);
        glVertex3f(4, 0, -4);
        glVertex3f(16, 0, -4);
        glVertex3f(16, 2, -4);
        glVertex3f(4, 2, -4);
        glEnd();
        glColor3f(0, 0, 0);
        for (k = 2; k < 18; k += 4)
        {
            glBegin(GL_QUADS);
            glVertex3f(k, 0, -0.1);
            glVertex3f(k, 4, -0.1);
            glVertex3f(k + 2, 4, -0.1);
            glVertex3f(k + 2, 0, -0.1);
            glEnd();
        }
        glPushMatrix();
        glTranslatef(20, 0, 0);
        glColor3f(1, 0.894, 0.709);
        glBegin(GL_QUADS);
        glVertex3f(0, 0, 0);
        glVertex3f(0, 0, -40);
        glVertex3f(-4, 0, -40);
        glVertex3f(-4, 0, 0);
        glEnd();
        glColor3f(1, 0.972, 0.862);
        glBegin(GL_QUADS);
        glVertex3f(-4, 0, -4);
        glVertex3f(-4, 0, -36);
        glVertex3f(-4, 2, -36);
    glVertex3f(-4, 2, -4);
    glEnd();
    glColor3f(0, 0, 0);
    for (k = -2; k > -38; k -= 6)
    {
        glBegin(GL_QUADS);
        glVertex3f(-0.1, 0, k);
        glVertex3f(-0.1, 4, k);
        glVertex3f(-0.1, 4, k - 2);
        glVertex3f(-0.1, 0, k - 2);
        glEnd();
    }
    glTranslatef(0, 0, -40);
    glColor3f(1, 0.894, 0.709);
    glBegin(GL_QUADS);
    glVertex3f(0, 0, 0);
    glVertex3f(-20, 0, 0);
    glVertex3f(-20, 0, 4);
    glVertex3f(0, 0, 4);
    glEnd();
    glColor3f(1, 0.972, 0.862);
    glBegin(GL_QUADS);
    glVertex3f(-4, 0, 4);
    glVertex3f(-16, 0, 4);
    glVertex3f(-16, 2, 4);
    glVertex3f(-4, 2, 4);
    glEnd();
    glColor3f(0, 0, 0);
    for (k = -2; k > -18; k -= 4)
    {
        glBegin(GL_QUADS);
        glVertex3f(k, 0, 0.1);
        glVertex3f(k, 4, 0.1);
        glVertex3f(k - 2, 4, 0.1);
        glVertex3f(k - 2, 0, 0.1);
        glEnd();
    }
    glPopMatrix();
    glColor3f(1, 0.894, 0.709);
    glBegin(GL_QUADS);
    glVertex3f(0, 0, 0);
    glVertex3f(0, 0, -40);
    glVertex3f(4, 0, -40);
    glVertex3f(4, 0, 0);
    glEnd();
```

```
        glColor3f(1, 0.972, 0.862);
        glBegin(GL_QUADS);
        glVertex3f(4, 0, -4);
        glVertex3f(4, 0, -36);
        glVertex3f(4, 2, -36);
        glVertex3f(4, 2, -4);
        glEnd();
        glColor3f(0, 0, 0);
        for (k = -2; k > -38; k -= 6)
        {
            glBegin(GL_QUADS);
            glVertex3f(0.1, 0, k);
            glVertex3f(0.1, 4, k);
            glVertex3f(0.1, 4, k - 2);
            glVertex3f(0.1, 0, k - 2);
            glEnd();
        }
        glPopMatrix();
    }
}


class temple //construction of temple
{
    float stair[4][3];
    float room[8][3];
    float ceil[6][3];

public:
    temple()
    {
        stair[0][0] = 0;
        stair[0][1] = 0;
        stair[0][2] = 0;
        stair[1][0] = 12;
        stair[1][1] = 0;
        stair[1][2] = 0;
        stair[2][0] = 12;
        stair[2][1] = 0;
        stair[2][2] = -12;
        stair[3][0] = 0;
        stair[3][1] = 0;
        stair[3][2] = -12;

        room[0][0] = 0;
    room[0][1] = 0;
    room[0][2] = 0;
    room[1][0] = 0;
    room[1][1] = 6;
    room[1][2] = 0;
    room[2][0] = 0;
    room[2][1] = 6;
    room[2][2] = -7;
    room[3][0] = 0;
    room[3][1] = 0;
    room[3][2] = -7;
    room[4][0] = 7;
    room[4][1] = 0;
    room[4][2] = -7;
    room[5][0] = 7;
    room[5][1] = 6;
    room[5][2] = -7;
    room[6][0] = 7;
    room[6][1] = 6;
    room[6][2] = 0;
    room[7][0] = 7;
    room[7][1] = 0;
    room[7][2] = 0;

    ceil[0][0] = 0;
    ceil[0][1] = 6;
    ceil[0][2] = 4;
    ceil[1][0] = 3.5;
    ceil[1][1] = 9;
    ceil[1][2] = 4;
    ceil[2][0] = 7;
    ceil[2][1] = 6;
    ceil[2][2] = 4;
    ceil[3][0] = 7;
    ceil[3][1] = 6;
    ceil[3][2] = -9;
    ceil[4][0] = 3.5;
    ceil[4][1] = 9;
    ceil[4][2] = -9;
    ceil[5][0] = 0;
    ceil[5][1] = 6;
    ceil[5][2] = -9;
}
void disp_stair(int x, int y, int z)
{
    glColor3f(1, 0.960, 0.933);
    glBegin(GL_QUADS);
    glVertex3f(stair[0][0] - x, stair[0][1] - y, stair[0][2] + z);
    glVertex3f(stair[1][0] + x, stair[1][1] - y, stair[1][2] + z);
    glVertex3f(stair[2][0] + x, stair[2][1] - y, stair[2][2] - z);
    glVertex3f(stair[3][0] - x, stair[3][1] - y, stair[3][2] - z);
    glEnd();
    glColor3f(0.933, 0.913, 0.8);
    glBegin(GL_QUADS);
    glVertex3f(stair[0][0] - x, stair[0][1] - y, stair[0][2] + z);
    glVertex3f(stair[0][0] - x, stair[0][1] - 1 - y, stair[0][2] + z);
    glVertex3f(stair[1][0] + x, stair[1][1] - 1 - y, stair[1][2] + z);
    glVertex3f(stair[1][0] + x, stair[1][1] - y, stair[1][2] + z);

    glVertex3f(stair[1][0] + x, stair[1][1] - y, stair[1][2] + z);
    glVertex3f(stair[1][0] + x, stair[1][1] - 1 - y, stair[1][2] + z);
    glVertex3f(stair[2][0] + x, stair[2][1] - 1 - y, stair[2][2] - z);
    glVertex3f(stair[2][0] + x, stair[2][1] - y, stair[2][2] - z);

    glVertex3f(stair[2][0] + x, stair[2][1] - y, stair[2][2] - z);
    glVertex3f(stair[3][0] - x, stair[3][1] - y, stair[3][2] - z);
    glVertex3f(stair[3][0] - x, stair[3][1] - 1 - y, stair[3][2] - z);
    glVertex3f(stair[2][0] + x, stair[2][1] - 1 - y, stair[2][2] - z);

    glVertex3f(stair[3][0] - x, stair[3][1] - y, stair[3][2] - z);
    glVertex3f(stair[0][0] - x, stair[0][1] - y, stair[0][2] + z);
    glVertex3f(stair[0][0] - x, stair[0][1] - 1 - y, stair[0][2] + z);
    glVertex3f(stair[3][0] - x, stair[3][1] - 1 - y, stair[3][2] - z);
    glEnd();
}

void disp_room()
{
    glColor3f(0.803, 0.803, 0.756);
    glBegin(GL_QUADS);
    glVertex3fv(room[0]);
    glVertex3fv(room[1]);
    glVertex3fv(room[2]);
    glVertex3fv(room[3]);
    glVertex3fv(room[3]);
    glVertex3fv(room[2]);
    glVertex3fv(room[5]);
    glVertex3fv(room[4]);
    glVertex3fv(room[4]);
    glVertex3fv(room[5]);
    glVertex3fv(room[6]);
    glVertex3fv(room[7]);
    glVertex3fv(room[1]);
    glVertex3fv(room[2]);
    glVertex3fv(room[5]);
    glVertex3fv(room[6]);
    glVertex3fv(room[0]);
    glVertex3f(room[0][0] + 1, room[0][1], room[0][2]);
    glVertex3f(room[1][0] + 1, room[1][1], room[1][2]);
    glVertex3fv(room[1]);
    glVertex3fv(room[7]);
    glVertex3f(room[7][0] - 1, room[7][1], room[7][2]);
    glVertex3f(room[6][0] - 1, room[6][1], room[6][2]);
    glVertex3fv(room[6]);
    glEnd();
}
void disp_ceil()
{
    glColor3f(1, 0.843, 0);
    glBegin(GL_TRIANGLES);
    glVertex3fv(ceil[2]);
    glVertex3fv(ceil[1]);
    glVertex3fv(ceil[0]);
    glVertex3fv(ceil[3]);
    glVertex3fv(ceil[4]);
    glVertex3fv(ceil[5]);
    glEnd();
    glColor3f(0.933, 0.866, 0.509);
    glBegin(GL_POLYGON);
    glVertex3fv(ceil[2]);
    glVertex3fv(ceil[1]);
    glVertex3fv(ceil[4]);
    glVertex3fv(ceil[3]);
    glEnd();
    glBegin(GL_POLYGON);
    glVertex3fv(ceil[0]);
    glVertex3fv(ceil[1]);
    glVertex3fv(ceil[4]);
    glVertex3fv(ceil[5]);
    glEnd();
}
void draw_pil()
{
    GLUquadricObj *quadratic;
    quadratic = gluNewQuadric();
    glPushMatrix();
    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
    glColor3f(0.933, 0.866, 0.509);
    gluCylinder(quadratic, 0.5, 0.5, 6.0f, 32, 32);
    glPopMatrix();
}
void draw_mesh()
{
    glColor3f(1, 0.843, 0);
    for (float i = 0; i < 0.9; i += 0.2)
        for (float j = 0; j < 6; j += 0.2)
        {
            glBegin(GL_LINE_LOOP);
            glVertex3f(i, j, 0);
            glVertex3f(i + 0.2, j, 0);
            glVertex3f(i + 0.2, j + 0.2, 0);
            glVertex3f(i, j + 0.2, 0);
            glEnd();
        }
}
void disp_temple()
{
    disp_stair(0, 0, 0);
    glPushMatrix();
    disp_stair(2, 1, 2);
    disp_stair(4, 2, 4);

    glPopMatrix();
    glPushMatrix();
    glTranslatef(4, 0, -9);
    glRotatef(-90, 0, 1, 0);
    disp_room();
    disp_ceil();
    glPushMatrix();
    glTranslatef(0.4, 0, 2.5);
    draw_pil();
    glTranslatef(6.2, 0, 0);
    draw_pil();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(1.5, 0, -3);
    draw_idol();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(1, 0, 0);
```

```
        draw_mesh();
        glTranslatef(4, 0, 0);
        draw_mesh();
        glPopMatrix();
        glPopMatrix();
    }

} temp;

class building //construction of the block buildings
{
    float structure[8][3];

public:
    building(float a, float b, float c)
    {
        structure[0][0] = 0;
        structure[0][1] = 0;
        structure[0][2] = 0;
        structure[1][0] = a;
        structure[1][1] = 0;
        structure[1][2] = 0;
        structure[2][0] = a;
        structure[2][1] = b;
        structure[2][2] = 0;
        structure[3][0] = 0;
        structure[3][1] = b;
        structure[3][2] = 0;
        structure[4][0] = 0;
        structure[4][1] = 0;
        structure[4][2] = c;
        structure[5][0] = a;
        structure[5][1] = 0;
        structure[5][2] = c;
        structure[6][0] = a;
        structure[6][1] = b;
        structure[6][2] = c;
        structure[7][0] = 0;
        structure[7][1] = b;
        structure[7][2] = c;
    }
    void disp_build(char text[15], char side = '/0')
    {
        float door[3];
        glColor3f(1, 0.980, 0.980);
    // Faces of the building
    glBegin(GL_QUADS);
    glVertex3fv(structure[0]);
    glVertex3fv(structure[1]);
    glVertex3fv(structure[2]);
    glVertex3fv(structure[3]);
    glEnd();
    glBegin(GL_QUADS);
    glVertex3fv(structure[0]);
    glVertex3fv(structure[4]);
    glVertex3fv(structure[7]);
    glVertex3fv(structure[3]);
    glEnd();
    glBegin(GL_QUADS);
    glVertex3fv(structure[4]);
    glVertex3fv(structure[5]);
    glVertex3fv(structure[6]);
    glVertex3fv(structure[7]);
    glEnd();
    glBegin(GL_QUADS);
    glVertex3fv(structure[1]);
    glVertex3fv(structure[2]);
    glVertex3fv(structure[6]);
    glVertex3fv(structure[5]);
    glEnd();

    if (structure[1][0] > (-1 * structure[4][2]))
    {
        // Windows
        for (float i = 10; i < structure[2][1]; i += 10)
        {
            glPushMatrix();
            glTranslatef(0, i, 0);
            for (float j = 5; j < structure[1][0]; j += 15)
            {
                glColor3f(0, 0, 0);
                glBegin(GL_POLYGON);
                glVertex3f(j, 0, 0.1);
                glVertex3f(j + 5, 0, 0.1);
                glVertex3f(j + 5, 5, 0.1);
                glVertex3f(j, 5, 0.1);
                glEnd();
                glBegin(GL_POLYGON);
                glVertex3f(j, 0, structure[4][2] - 0.1);
            glVertex3f(j + 5, 0, structure[4][2] - 0.1);
            glVertex3f(j + 5, 5, structure[4][2] - 0.1);
            glVertex3f(j, 5, structure[4][2] - 0.1);
            glEnd();
        }
        for (float j = 0; j < structure[1][0]; j += 15)
        {
            glColor3f(1, 0, 0);
            glBegin(GL_POLYGON);
            glVertex3f(j, -10, 0.1);
            glVertex3f(j + 2, -10, 0.1);
            glVertex3f(j + 2, 10, 0.1);
            glVertex3f(j, 10, 0.1);
            glEnd();
            glBegin(GL_POLYGON);
            glVertex3f(j, -10, structure[4][2] - 0.1);
            glVertex3f(j + 2, -10, structure[4][2] - 0.1);
            glVertex3f(j + 2, 10, structure[4][2] - 0.1);
            glVertex3f(j, 10, structure[4][2] - 0.1);
            glEnd();
        }
        glPopMatrix();
    }

    // Door
    glColor3f(0, 0, 0);
    door[0] = (structure[1][0] / 2);
    glBegin(GL_POLYGON);
    glVertex3f(door[0] - 4, 0, 0.2);
    glVertex3f(door[0] + 4, 0, 0.2);
    glVertex3f(door[0] + 4, 7, 0.2);
    glVertex3f(door[0] - 4, 7, 0.2);
    glEnd();
    glPushMatrix();
    glTranslatef(10, 0, 3);
    draw_board();
    glPushMatrix();
    glTranslatef(2.75, 2.25, 0.1);
    glScalef(.01, .01, .01);
    glLineWidth(2);
    glColor3f(0, 0, 0);
    for (int c = 0; text[c] != 0; ++c)
        glutStrokeCharacter(GLUT_STROKE_ROMAN, text[c]);
    glPopMatrix();
    glPopMatrix();
```

```
        }
        else
        {
            for (float i = 10; i < structure[2][1]; i += 10)
            {
                glPushMatrix();
                glTranslatef(0, i, 0);
                for (float j = -5; j > structure[4][2]; j -= 15)
                {
                    glColor3f(0, 0, 0);
                    glBegin(GL_POLYGON);
                    glVertex3f(-0.1, 0, j);
                    glVertex3f(-0.1, 0, j - 5);
                    glVertex3f(-0.1, 5, j - 5);
                    glVertex3f(-0.1, 5, j);
                    glEnd();
                    glBegin(GL_POLYGON);
                    glVertex3f(structure[1][0] + 0.1, 0, j);
                    glVertex3f(structure[1][0] + 0.1, 0, j - 5);
                    glVertex3f(structure[1][0] + 0.1, 5, j - 5);
                    glVertex3f(structure[1][0] + 0.1, 5, j);
                    glEnd();
                }
                for (float j = 0; j > structure[4][2]; j -= 15)
                {
                    glColor3f(1, 0, 0);
                    glBegin(GL_POLYGON);
                    glVertex3f(-0.1, -10, j);
                    glVertex3f(-0.1, -10, j - 2);
                    glVertex3f(-0.1, 10, j - 2);
                    glVertex3f(-0.1, 10, j);
                    glEnd();
                    glBegin(GL_POLYGON);
                    glVertex3f(structure[1][0] + 0.1, -10, j);
                    glVertex3f(structure[1][0] + 0.1, -10, j - 2);
                    glVertex3f(structure[1][0] + 0.1, 10, j - 2);
                    glVertex3f(structure[1][0] + 0.1, 10, j);
                    glEnd();
                }
                glPopMatrix();
            }
            door[2] = (structure[4][2] / 2);
            door[0] = structure[1][0];
            glColor3f(0, 0, 0);
            if (side == 'r')
        {
            glBegin(GL_POLYGON);
            glVertex3f(door[0] + 0.2, 0, door[2] - 4);
            glVertex3f(door[0] + 0.2, 0, door[2] + 4);
            glVertex3f(door[0] + 0.2, 7, door[2] + 4);
            glVertex3f(door[0] + 0.2, 7, door[2] - 4);
            glEnd();
            glPushMatrix();
            glTranslatef(door[0] + 3, 0, -2);
            glRotatef(90, 0, 1, 0);
            draw_board();
            glPushMatrix();
            glTranslatef(1, 2, 0.1);
            glScalef(.01, .01, .01);
            glLineWidth(2);
            glColor3f(0, 0, 0);
            for (int c = 0; text[c] != 0; ++c)
                glutStrokeCharacter(GLUT_STROKE_ROMAN, text[c]);
            glPopMatrix();
            glPopMatrix();
        }
        else if (side == 'l')
        {
            glBegin(GL_POLYGON);
            glVertex3f(-0.2, 0, door[2] - 4);
            glVertex3f(-0.2, 0, door[2] + 4);
            glVertex3f(-0.2, 7, door[2] + 4);
            glVertex3f(-0.2, 7, door[2] - 4);
            glEnd();
            glPushMatrix();
            glTranslatef(-3, 0, -10);
            glRotatef(-90, 0, 1, 0);
            draw_board();
            glPushMatrix();
            glTranslatef(1, 2, 0.1);
            glScalef(.01, .01, .01);
            glLineWidth(2);
            glColor3f(0, 0, 0);
            for (int c = 0; text[c] != 0; ++c)
                glutStrokeCharacter(GLUT_STROKE_ROMAN, text[c]);
            glPopMatrix();
            glPopMatrix();
        }
    }
    glPushMatrix();
```

```
        glTranslatef(0, 10, 0);
        glColor3f(0, 0, 1);
        for (int i = 0; i < structure[2][1] - 5; i += 5)
        {
            glBegin(GL_LINES);
            glVertex3f(structure[0][0], i, structure[0][2] + 0.1);
            glVertex3f(structure[1][0], i, structure[0][2] + 0.1);
            glVertex3f(structure[0][0] - 0.1, i, structure[0][2]);
            glVertex3f(structure[0][0] - 0.1, i, structure[4][2]);
            glVertex3f(structure[4][0], i, structure[4][2] - 0.1);
            glVertex3f(structure[5][0], i, structure[4][2] - 0.1);
            glVertex3f(structure[5][0] + 0.1, i, structure[5][2]);
            glVertex3f(structure[1][0] + 0.1, i, structure[1][2]);

            glEnd();
        }
        glPopMatrix();
    }
};
building canteen(20, 30, -30);
building mech(20, 40, -40);
building chem(20, 40, -40);
building admin(40, 30, -20);
building ec(40, 30, -30);
building cs(30, 40, -40);


// Basketball Basket
void loop(float x, float y, float z)
{
    float xx, zz, d;
    glColor3f(1, 0, 0);
    glPointSize(2);
    glBegin(GL_POINTS);
    for (int i = 0; i < 360; i++)
    {
        d = i * (180 / 3.14);
        xx = cos(d) + x;
        zz = sin(d) + z;
        glVertex3f(xx, y, zz);
    }
    glEnd();
}

// Basketball Court
```

```
class bball
{
   float bordr[4][3];
   float bskt[8][3];

public:
   bball()
   {
       bordr[0][0] = 0;
       bordr[0][1] = 0;
       bordr[0][2] = 0;
       bordr[1][0] = 20;
       bordr[1][1] = 0;
       bordr[1][2] = 0;
       bordr[2][0] = 20;
       bordr[2][1] = 0;
       bordr[2][2] = -20;
       bordr[3][0] = 0;
       bordr[3][1] = 0;
       bordr[3][2] = -20;
       bskt[0][0] = 14;
       bskt[0][1] = 4.5;
       bskt[0][2] = -0.1;
       bskt[1][0] = 16;
       bskt[1][1] = 4.5;
       bskt[1][2] = -0.1;
       bskt[2][0] = 16;
       bskt[2][1] = 6.5;
       bskt[2][2] = -0.1;
       bskt[3][0] = 14;
       bskt[3][1] = 6.5;
       bskt[3][2] = -0.1;
       bskt[4][0] = 14, bskt[4][1] = 4.5;
       bskt[4][2] = -19.9;
       bskt[5][0] = 16;
       bskt[5][1] = 4.5;
       bskt[5][2] = -19.9;
       bskt[6][0] = 16;
       bskt[6][1] = 6.5;
       bskt[6][2] = -19.9;
       bskt[7][0] = 14;
       bskt[7][1] = 6.5;
       bskt[7][2] = -19.9;
   }
   void disp_court()
```

```
    {
        glPushMatrix();
        glTranslatef(0, 0.1, 0);
        glColor3f(0.745, 0.745, 0.745);
        glBegin(GL_QUADS);
        glVertex3fv(bordr[0]);
        glVertex3fv(bordr[1]);
        glVertex3fv(bordr[2]);
        glVertex3fv(bordr[3]);
        glEnd();
        glColor3f(1, 0.270, 0);
        GLUquadricObj *quadratic;
        quadratic = gluNewQuadric();
        GLUquadricObj *quadratic1;
        quadratic1 = gluNewQuadric();
        glPushMatrix();
        glTranslatef(15, 0, 0);
        loop(0, 5, -1);
        glPushMatrix();
        glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
        glColor3f(0.698, 0.133, 0.133);
        gluCylinder(quadratic, 0.1, 0.1, 5.0f, 32, 32);
        glPopMatrix();
        glPopMatrix();
        glPushMatrix();
        glTranslatef(15, 0, -20);
        loop(0, 5, 1);
        glPushMatrix();
        glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
        glColor3f(0.698, 0.133, 0.133);
        gluCylinder(quadratic1, 0.1, 0.1, 5.0f, 32, 32);
        glPopMatrix();
        glPopMatrix();
        glColor3f(0.745, 0.745, 0.745);
        glBegin(GL_QUADS);
        for (int i = 0; i < 8; i++)
            glVertex3fv(bskt[i]);
        glEnd();
        glPopMatrix();
    }
};
bball crt1;

// Football Ground
class ground
```

```
{
    float bordr[4][3];

public:
    ground()
    {
        bordr[0][0] = 0;
        bordr[0][1] = -2;
        bordr[0][2] = 0;
        bordr[1][0] = 40;
        bordr[1][1] = -2;
        bordr[1][2] = 0;
        bordr[2][0] = 40;
        bordr[2][1] = -2;
        bordr[2][2] = -40;
        bordr[3][0] = 0;
        bordr[3][1] = -2;
        bordr[3][2] = -40;
    }
    void ground_disp()
    {
        float t = 6;
        for (int i = 0; i < 3; i++, t -= 2)
        {
            glPushMatrix();
            glTranslatef(0, i, 0);
            glColor3f(0.803, 0.701, 0.545);
            glBegin(GL_POLYGON);
            glVertex3f(bordr[0][0], bordr[0][1], bordr[0][2]);
            glVertex3f(bordr[1][0], bordr[1][1], bordr[1][2]);
            glVertex3f(bordr[1][0], bordr[1][1], bordr[1][2] - t);
            glVertex3f(bordr[0][0], bordr[0][1], bordr[0][2] - t);
            glEnd();
            glColor3f(0.545, 0.474, 0.368);
            glBegin(GL_POLYGON);
            glVertex3f(bordr[0][0], bordr[0][1], bordr[0][2] - t);
            glVertex3f(bordr[1][0], bordr[1][1], bordr[1][2] - t);
            glVertex3f(bordr[1][0], bordr[1][1] - 1, bordr[1][2] - t);
            glVertex3f(bordr[0][0], bordr[0][1] - 1, bordr[0][2] - t);
            glEnd();
            glColor3f(0.803, 0.701, 0.545);
            glBegin(GL_POLYGON);
            glVertex3f(bordr[1][0], bordr[1][1], bordr[1][2]);
            glVertex3f(bordr[2][0], bordr[2][1], bordr[2][2]);
            glVertex3f(bordr[2][0] - t, bordr[2][1], bordr[2][2]);
            glVertex3f(bordr[1][0] - t, bordr[1][1], bordr[1][2]);
            glEnd();
            glColor3f(0.545, 0.474, 0.368);
            glBegin(GL_POLYGON);
            glVertex3f(bordr[1][0] - t, bordr[1][1], bordr[1][2]);
            glVertex3f(bordr[2][0] - t, bordr[2][1], bordr[2][2]);
            glVertex3f(bordr[2][0] - t, bordr[2][1] - 1, bordr[2][2]);
            glVertex3f(bordr[1][0] - t, bordr[1][1] - 1, bordr[1][2]);
            glEnd();
            glColor3f(0.803, 0.701, 0.545);
            glBegin(GL_POLYGON);
            glVertex3f(bordr[2][0], bordr[2][1], bordr[2][2]);
            glVertex3f(bordr[3][0], bordr[3][1], bordr[3][2]);
            glVertex3f(bordr[3][0], bordr[3][1], bordr[3][2] + t);
            glVertex3f(bordr[2][0], bordr[2][1], bordr[2][2] + t);
            glEnd();
            glColor3f(0.545, 0.474, 0.368);
            glBegin(GL_POLYGON);
            glVertex3f(bordr[2][0], bordr[2][1], bordr[2][2] + t);
            glVertex3f(bordr[3][0], bordr[3][1], bordr[3][2] + t);
            glVertex3f(bordr[3][0], bordr[3][1] - 1, bordr[3][2] + t);
            glVertex3f(bordr[2][0], bordr[2][1] - 1, bordr[2][2] + t);
            glEnd();
            glColor3f(0.803, 0.701, 0.545);
            glBegin(GL_POLYGON);
            glVertex3f(bordr[3][0], bordr[3][1], bordr[3][2]);
            glVertex3f(bordr[0][0], bordr[0][1], bordr[0][2]);
            glVertex3f(bordr[0][0] + t, bordr[0][1], bordr[0][2]);
            glVertex3f(bordr[3][0] + t, bordr[3][1], bordr[3][2]);
            glEnd();
            glColor3f(0.545, 0.474, 0.368);
            glBegin(GL_POLYGON);
            glVertex3f(bordr[0][0] + t, bordr[0][1], bordr[0][2]);
            glVertex3f(bordr[3][0] + t, bordr[3][1], bordr[3][2]);
            glVertex3f(bordr[3][0] + t, bordr[3][1] - 1, bordr[3][2]);
            glVertex3f(bordr[0][0] + t, bordr[0][1] - 1, bordr[0][2]);
            glEnd();
            glPopMatrix();
        }
        glPushMatrix();
        glTranslatef(16.5, -3, -7);
        glColor3f(0.827, 0.827, 0.827);
        glLineWidth(10);
        glBegin(GL_LINE_LOOP);
        glVertex3f(0, 0, 0);
```

```
        glVertex3f(0, 2, 0);
        glVertex3f(4, 2, 0);
        glVertex3f(4, 0, 0);
        glEnd();
        glPopMatrix();
        glPushMatrix();
        glTranslatef(16.5, -3, -33);
        glColor3f(0.827, 0.827, 0.827);
        glLineWidth(10);
        glBegin(GL_LINE_LOOP);
        glVertex3f(0, 0, 0);
        glVertex3f(0, 2, 0);
        glVertex3f(4, 2, 0);
        glVertex3f(4, 0, 0);
        glEnd();
        glPopMatrix();
    }
} fball;

// Modifies Aspect Ratio and Camera Perspective
void changeSize(int w, int h)
{
    float ratio = ((float)w) / ((float)h);    // Window Aspect Ratio
    glMatrixMode(GL_PROJECTION);              // Projection matrix is active
    glLoadIdentity();                         // Reset the projection
    gluPerspective(100.0, ratio, 0.1, 100.0); // Perspective transformation
    glMatrixMode(GL_MODELVIEW);               // Return to modelview mode
    glViewport(0, 0, w, h);                   // Set viewport (drawing area) to entire
window
}

// Updates Camera Position
void update(void)
{
    if (deltaMove)
    { // update camera position
        x += deltaMove * lx * 0.38;
        z += deltaMove * lz * 0.38;
    }else if(deltaMoveCross){
        x += -deltaMoveCross*lz*0.38;
        z += deltaMoveCross*lx*0.38;
    }
    if (vertmove == 1)
        y += 0.1;
    if (vertmove == -1)
```

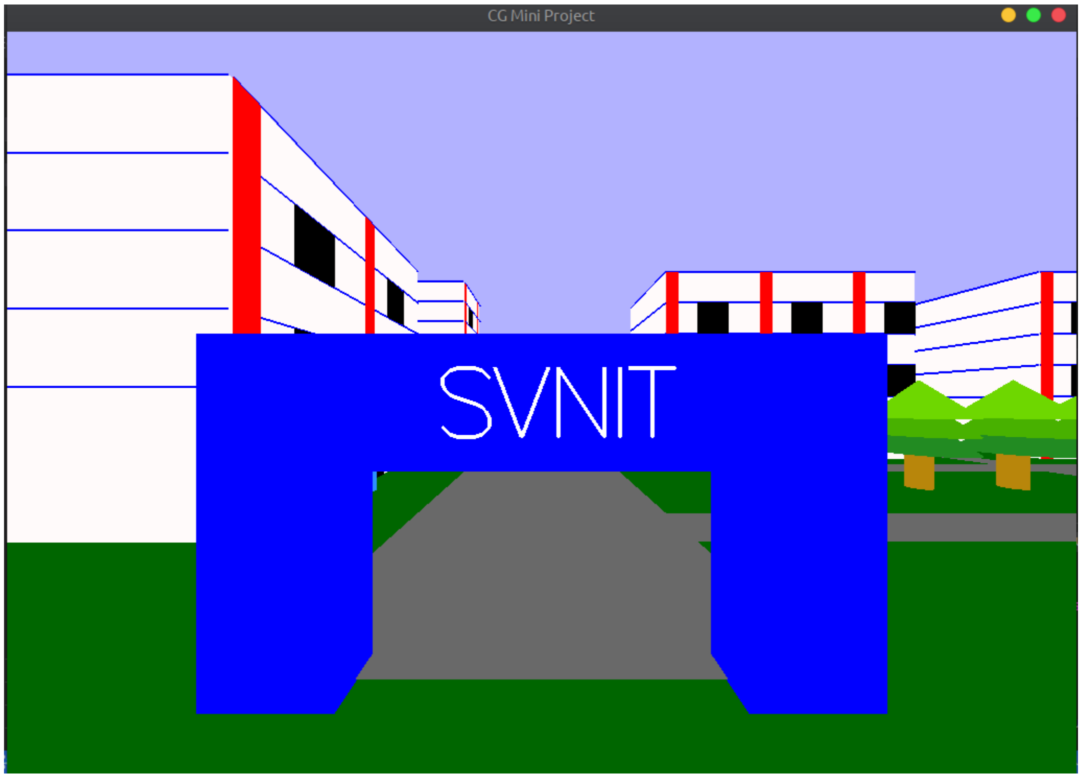
```
       y -= 0.1;
   restrict();
   glutPostRedisplay(); // redisplay everything
}

// Display Roads
void disp_roads()
{
   glColor3f(0.411, 0.411, 0.411);
   // Front Road Towards Z Axis
   glBegin(GL_QUADS);
   glVertex3f(-40, 0.1, 90);
   glVertex3f(-40, 0.1, -70);
   glVertex3f(-20, 0.1, -70);
   glVertex3f(-20, 0.1, 90);
   glEnd();

   // Side Road
   glBegin(GL_QUADS);
   glVertex3f(-20, 0.1, 55);
   glVertex3f(90, 0.1, 55);
   glVertex3f(90, 0.1, 60);
   glVertex3f(-20, 0.1, 60);
   glEnd();

   //Side Road
   glBegin(GL_QUADS);
   glVertex3f(-20, 0.1, 75);
   glVertex3f(40, 0.1, 75);
   glVertex3f(40, 0.1, 80);
   glVertex3f(-20, 0.1, 80);
   glEnd();

   //Front Road
   glBegin(GL_QUADS);
   glVertex3f(35, 0.1, 75);
   glVertex3f(35, 0.1, -70);
   glVertex3f(40, 0.1, -70);
   glVertex3f(40, 0.1, 75);
   glEnd();
}

// Draws Trees
void trees()
{
   GLUquadricObj *quadratic;
   GLUquadricObj *quadratic1;
   quadratic1 = gluNewQuadric();
   quadratic = gluNewQuadric();
   glPushMatrix();
   glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
   glColor3f(0.721, 0.525, 0.043);
   gluCylinder(quadratic, 1, 1, 10.0f, 32, 32);
   glPopMatrix();
   glTranslatef(0, 2, 0);
   glPushMatrix();
   float k = 0;
   for (int i = 0, j = 0; i < 3; i++, j += 0.5, k += 0.15)
   {
       glTranslatef(0, 1.8, 0);
       glColor3f(0.133 + k, 0.545 + k, 0.133 - k);
       glPushMatrix();
       glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
       gluCylinder(quadratic1, 4 - j, 0, 4.0f, 32, 32);
       glPopMatrix();
   }
   glPopMatrix();
}

// Draw the Main Gate
void draw_arch(char text[5])
{
   glColor3f(0, 0, 1);
   glPushMatrix();
   glTranslatef(0, 3.5, 0);
   glScalef(4, 7, 2);
   glutSolidCube(1);
   glPopMatrix();
   glPushMatrix();
   glTranslatef(16, 3.5, 0);
   glScalef(4, 7, 2);
   glutSolidCube(1);
   glPopMatrix();
   glPushMatrix();
   glTranslatef(8, 9, 0);
   glScalef(20, 4, 2);
   glutSolidCube(1);
   glPopMatrix();
   glPushMatrix();
   glTranslatef(5, 8, 1.1);
```

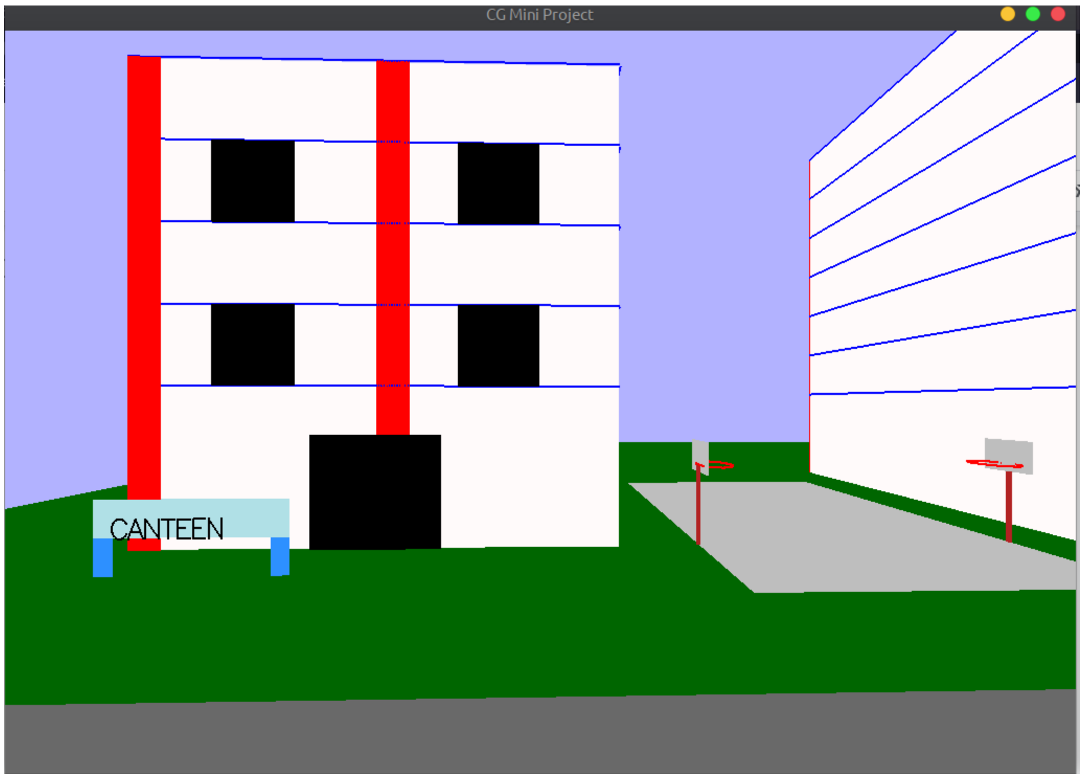
```
    glScalef(.02, .02, .02);
    glLineWidth(4.5);
    glColor3f(1, 1, 1);
    for (int c = 0; text[c] != 0; ++c)
        glutStrokeCharacter(GLUT_STROKE_ROMAN, text[c]);
    glPopMatrix();
}

// Display Function
void display()
{
    glClearColor(0.7, 0.7, 1, 0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glLoadIdentity();
    gluLookAt(
        x, y, z,
        x + lx, y, z + lz,
        0.0, 1.0, 0.0);

    // Green Ground
    glColor3f(0, 0.4, 0);
    glBegin(GL_QUADS);
    glVertex3f(-100, 0, 100);
    glVertex3f(100, 0, 100);
    glVertex3f(100, 0, 20);
    glVertex3f(-100, 0, 20);
    glVertex3f(-100, 0, 20);
    glVertex3f(-15, 0, 20);
    glVertex3f(-15, 0, -100);
    glVertex3f(-100, 0, -100);
    glVertex3f(-15, 0, -100);
    glVertex3f(100, 0, -100);
    glVertex3f(100, 0, -20);
    glVertex3f(-15, 0, -20);
    glVertex3f(25, 0, -20);
    glVertex3f(100, 0, -20);
    glVertex3f(100, 0, 20);
    glVertex3f(25, 0, 20);
    glEnd();

    // Draw Map
    draw_map();

    // Draw Roads
```

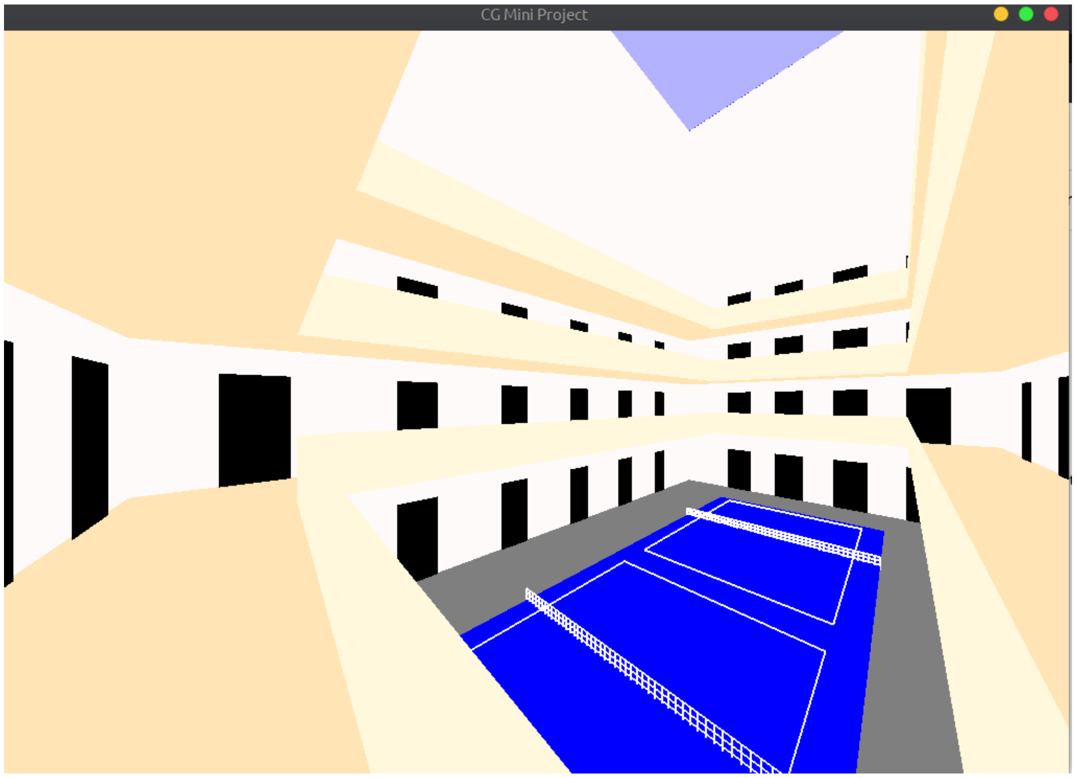
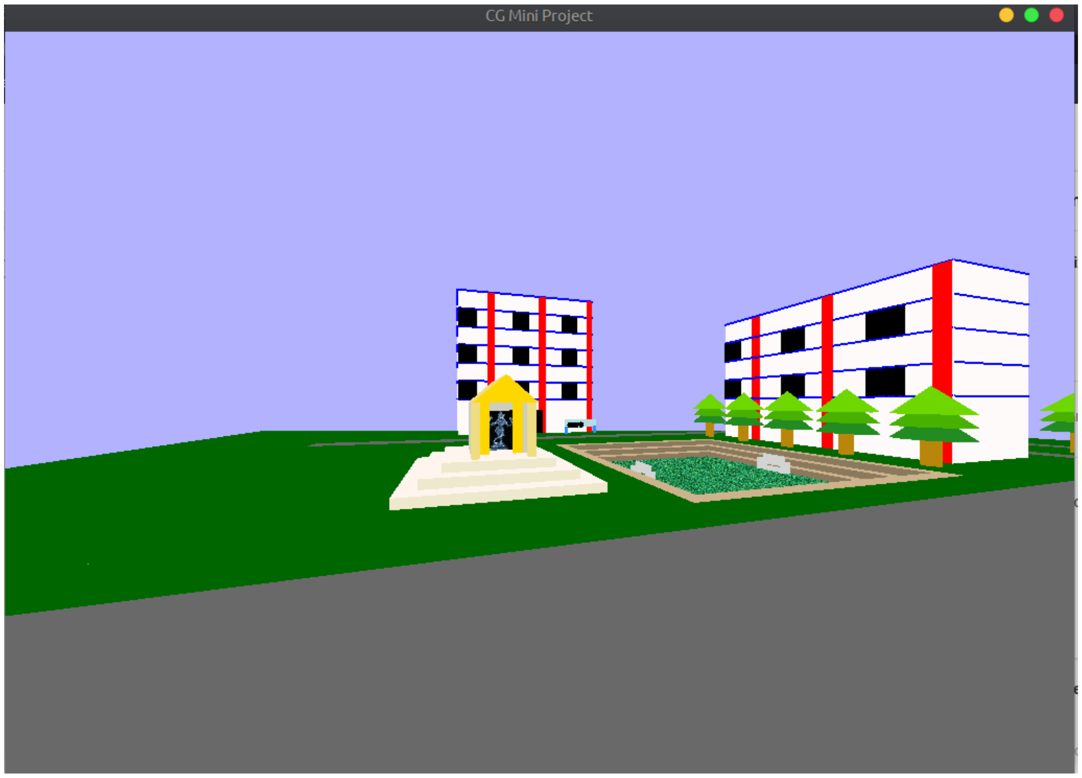
```
disp_roads();

// Draw Trees
for (int i = -10; i < 40; i += 10)
{
    glPushMatrix();
    glTranslatef(i, 0, 67);
    trees();
    glPopMatrix();
}
for (int i = 45; i < 90; i += 10)
{
    glPushMatrix();
    glTranslatef(i, 0, 65);
    trees();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(i, 0, 75);
    trees();
    glPopMatrix();
}
for (int i = -10; i < 35; i += 10)
{
    glPushMatrix();
    glTranslatef(i, 0, 25);
    trees();
    glPopMatrix();
}

// Draw Gate
glPushMatrix();
glTranslatef(-38, 0, 90);
draw_arch("SVNIT");
glPopMatrix();

// Draw Canteen Building
glPushMatrix();
glTranslatef(-70, 0, 80);
canteen.disp_build("CANTEEN", 'r');
glPopMatrix();

// Draw Basketball Court
glPushMatrix();
glTranslatef(-65, 0, 45);
crt1.disp_court();
```

```
    glPopMatrix();

    // Draw Mechanical Building
    glPushMatrix();
    glTranslatef(-70, 0, 20);
    mech.disp_build("MECH", 'r');
    glPopMatrix();

    // Draw Badminton Court inside Mech Building
    mech_court();

    // Draw Chemical Building
    glPushMatrix();
    glTranslatef(-70, 0, -30);
    chem.disp_build("CHEM", 'r');
    glPopMatrix();


    // Draw Admin Building
    glPushMatrix();
    glTranslatef(-10, 0, 50);
    admin.disp_build("ADMIN");
    glPopMatrix();

    // Draw Football Court
    glPushMatrix();
    glTranslatef(-15, 0, 20);
    fball.ground_disp();
    glPopMatrix();

    // Draw EC Building
    glPushMatrix();
    glTranslatef(50, 0, 50);
    ec.disp_build("EC");
    glPopMatrix();


    // Draw Comps Building
    glPushMatrix();
    glTranslatef(50, 0, 0);
    cs.disp_build("COMPS", 'l');
    glPopMatrix();


    // Draw
    glPushMatrix();
    glTranslatef(-6, 3, -30);
    temp.disp_temple();
    glPopMatrix();

    glutSwapBuffers();
    glFlush();
}

// Key Press Events
void pressKey(unsigned char key, int xx, int yy)
{
    switch (key)
    {
    case 'w':
        deltaMove = 1.0;
        glutIdleFunc(update);
        break;
    case 's':
        deltaMove = -1.0;
        glutIdleFunc(update);
        break;
    case 'a':
        deltaMoveCross = -1.0;
        glutIdleFunc(update);
        break;
    case 'd':
        deltaMoveCross = 1.0;
        glutIdleFunc(update);
        break;
    }

    switch((int)key){

    case 32:
        vertmove = toggle==0?1:-1;
        toggle = 1 - toggle;
        glutIdleFunc(update);
        break;
    }
}

// Key Release Events
void releaseKey(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'w':
        deltaMove = 0.0;
        glutIdleFunc(NULL);
        break;
    case 's':
        deltaMove = 0.0;
        glutIdleFunc(NULL);
        break;
    case 'a':
        deltaMoveCross = 0.0;
        glutIdleFunc(NULL);
        break;
    case 'd':
        deltaMoveCross = 0.0;
        glutIdleFunc(NULL);
        break;
    }
    switch((int)key){
    case 32:
        vertmove = 0;
        glutIdleFunc(update);
        break;
    }

}

// Mouse Drag Events
void mouseMove(int x, int y)
{
    if (isDragging)
    { // only when dragging
        // update the change in angle
        deltaAngle = (x - xDragStart) * -0.005;
        // camera's direction is set to angle + deltaAngle
        lx = sin(angle + deltaAngle);
        lz = -cos(angle + deltaAngle);
        glutIdleFunc(update);
    }
}

// Mouse Drag Start and Stop Events
void mouseButton(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON)
    {
        if (state == GLUT_DOWN)
        {
            isDragging = 1;
            xDragStart = x;
        }
        else
        {                        /* (state = GLUT_UP) */
            angle += deltaAngle; // update camera turning angle
            isDragging = 0;      // no longer dragging
            glutIdleFunc(NULL);
        }
    }
}

// Main Function Window Config
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(1024, 768);
    glutCreateWindow("CG Mini Project");
    glutReshapeFunc(changeSize); // window reshape callback
    glutDisplayFunc(display);

    glutIgnoreKeyRepeat(1);
    glutKeyboardFunc(pressKey);
    glutKeyboardUpFunc(releaseKey);
    glutMouseFunc(mouseButton); // process mouse button push/release
    glutMotionFunc(mouseMove);  // process mouse dragging motion
    glutMainLoop();
}
```

Screenshots:






CG Mini Project

CG Mini Project