**ABSTRACT**

OpenGL is widely used in many industrial application fields as a powerful 3D graphics library. Especially, it plays important roles in displaying the visualization emulation model of digital cores. In this project, we present a concept of a 3D interface to 3D Classroom. We argue the presentations of such kind are a lot more useful to demonstrate projects, conference talks, or computer graphics education tutorials and lectures with the results of 3D animation, effects, and others alongside with the presentation in place instead of switching between a regular presentation software to the demonstration and back – the demo and the presentation are combined together in one unit and can serve as an educational piece on its own.

**TABLE OF CONTENTS**

Abstract i

Acknowledgement ii

Table of Contents iii

List of Figures iv

**CHAPTER**  **PAGE NO.**

1. Introduction…………………………………………………………………1
   1. Computer Graphics
   2. OpenGL
2. Software Requirement Specification …………….………………..............5
   1. Hardware Requirement
   2. Software Requirement
   3. Functional Requirement
3. About the Project …………………………………………………………..6
   1. Overview
   2. Objective
   3. User Interface
   4. Built-in Functions
4. Implementation …………………………………………………………….9
   1. Source Code
5. Results …………… .…………………………………………………….22
6. Conclusion and Future Work………………………………………….......25
   1. Conclusion
   2. Future Enhancement
7. Bibliography ……………………………………………………………….26

**LIST OF FIGURES**

Fig 1.1: Components of Computer Graphics Architecture and their Working 2

Fig 1.2: OpenGL Library Organization 4

Fig 5.1: Cube view 29

Fig 5.2: Door view 29

Fig 5.3: Classroom front view 30

Fig 5.4: Classroom side view 30

Fig 5.5: Window view 31

Fig 5.6: Keyboard function 31

Fig 5.7: Light 1 off 32

Fig 5.8: Light 1 and 2 off 32

**Chapter 1**

**INTRODUCTION**

* 1. **Computer Graphics**

Computer Graphics involves technology to access. The Process transforms and presents information in a visual form. The role of computer graphics insensible. In today life, computer graphics has now become a common element in user interfaces, T.V. commercial motion pictures.

Computer Graphics is the creation of pictures with the help of a computer. The end product of the computer graphics is a picture it may be a business graph, drawing, and engineering.

In computer graphics, two or three-dimensional pictures can be created that are used for research. Many hardware devices algorithm has been developing for improving the speed of picture generation with the passes of time. It includes the creation storage of models and image of objects. These models for various fields like engineering, mathematical and so on.

Today computer graphics is entirely different from the earlier one. It is not possible. It is an interactive user can control the structure of an object of various input devices.

Computer graphics deals with generating [images](https://en.wikipedia.org/wiki/Image) with the aid of [co](https://en.wikipedia.org/wiki/Computers)mputers. Today, computer graphics is a core technology in digital photography, film, video games, cell phone and computer displays, and many specialized applications. A great deal of specialized hardware and software has been developed, with the displays of most devices being driven by [computer graphics hardware](https://en.wikipedia.org/wiki/Graphics_hardware). It is a vast and recently developed area of computer science. The phrase was coined in 1960 by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, or typically in the context of film as [computer generated imagery](https://en.wikipedia.org/wiki/Computer-generated_imagery) (CGI). The non-artistic aspects of computer graphics are the subject of [computer science](https://en.wikipedia.org/wiki/Computer_graphics_(computer_science)) research.

Some topics in computer graphics include user interface design, sprite graphics, rendering, ray tracing, geometry processing, computer animation, vector graphics, 3D modeling, shaders, GPU design, implicit surfaces, visualization, scientific computing, image processing, computational photography, scientific visualization, computational geometry and computer vision, among others. The overall methodology depends heavily on the underlying sciences of geometry, optics, physics, and perception.

Computer architecture can be made up of seven components:

1. Display processors
2. Pipeline architecture
3. The graphics pipeline
4. Vertex processing
5. Clipping and primitive assembly
6. Rasterization
7. Fragment processing

Graphics Architecture can be made up of seven components:

1. Display processors

2. Pipeline architectures

3. The graphics pipeline

4. Vertex processing

5. Clipping and primitive assembly

6. Rasterization

7. Fragment processing

Graphics Architecture can be made up of seven components:

1. Display processors

2. Pipeline architectures

3. The graphics pipeline

4. Vertex processing

5. Clipping and primitive assembly

6. Rasterization

7. Fragment processing

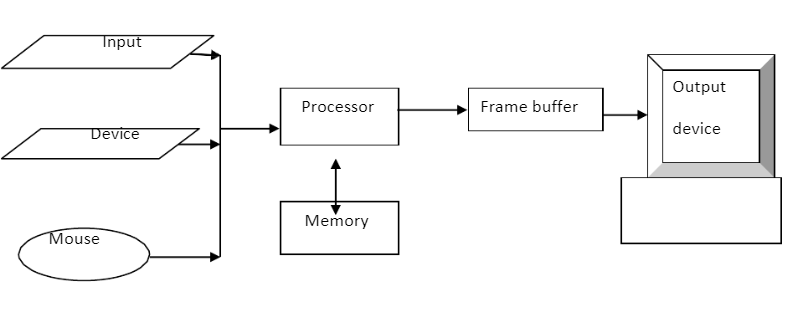


Fig 1.1: Components of Computer Graphics Architecture and their Working

Some applications of computer graphics are:

1. Display of information
2. Design
3. Simulation and animation
4. User interfaces

Computer graphics is responsible for displaying art and image data effectively and meaningfully to the consumer. It is also used for processing image data received from the physical world, such as photo and video content. Computer graphics development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, in general. and [computer vision](https://en.wikipedia.org/wiki/Computer_vision), among others. The overall methodology depends heavily on the underlying sciences of [geometry](https://en.wikipedia.org/wiki/Geometry), [optics](https://en.wikipedia.org/wiki/Optics), [physics](https://en.wikipedia.org/wiki/Physics), and [perception](https://en.wikipedia.org/wiki/Perception).

**1.2 OpenGL**

OpenGL (Open Graphics Library) is a cross-platform, hardware-accelerated, language-independent, industrial standard API for producing 3D (including 2D) graphics. Modern computers have dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering. OpenGL is the software interface to graphics hardware. In other words, OpenGL graphic rendering commands issued by your applications could be directed to the graphic hardware and accelerated.

We use 3 sets of libraries in our OpenGL programs:

1. Core OpenGL (GL): consists of hundreds of commands, which begin with a prefix "gl" (e.g., glColor, glVertex, glTranslate, glRotate). The Core OpenGL models an object via a set of geometric primitives such as point, line and polygon.
2. OpenGL Utility Library (GLU**)**: built on-top of the core OpenGL to provide important utilities (such as setting camera view and projection) and more building models (such as qradric surfaces and polygon tessellation). GLU commands start with a prefix "glu" (e.g., gluLookAt, gluPerspective).
3. OpenGL Utilities Toolkit (GLUT): OpenGL is designed to be independent of the windowing system or operating system. GLUT is needed to interact with the Operating System (such as creating a window, handling key and mouse inputs); it also provides more building models (such as sphere and torus). GLUT commands start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc). GLUT is platform independent, which is built on top of platform-specific OpenGL extension such as GLX for X Window System, WGL for Microsoft Window, and AGL, CGL or Cocoa for Mac OS.  
   Quoting from the [opengl.org](http://www.opengl.org/resources/libraries/glut/): "GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. GLUT is simple, easy, and small."  
   Alternative of GLUT includes SDL, ....
4. OpenGL Extension Wrangler Library (GLEW): "GLEW is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform.” A standalone utility called "glewinfo.exe" (under the "bin" directory) can be used to produce the list of OpenGL functions supported by your graphics system.

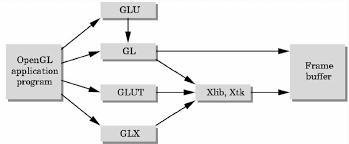


Fig 1.2: OpenGL Library Organization

**Chapter 2**

**SOFTWARE REQUIREMENT SPECIFICATION**

2.1 **Hardware Requirements**

Processor : Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz

Installed RAM : 8.00 GB (7.89 GB usable)

Keyboard : 100 keys

Mouse : Any pointing device

System type : 64-bit operating system, x64-based processor

2.2 **Software Requirements**

Operating System : Ubuntu **20.04.4 LTS/Visual studio code**

**Compiler : C compiler**

Framework used : OpenGL

Built using : C

2.3 **Functional Requirements**

* OpenGl APIs: If we want to have a control on the flow of the program and if we want to interact with the window system, then we use OpenGL APIs. Vertices are represented in the same manner internally, whether they are specified as two dimensional or three dimensional entities, everything that we do are here will be equally valid in three dimensions. Although OpenGL is easy to learn, compared with other APIs, it is nevertheless powerful. It supports the simple three dimensional programs and also supports the advanced rendering techniques.
* GL/glut.h: We use a readily available library called the OpenGL Utility Toolkit (GLUT) which provides the minimum functionality that should be expected in any modern windowing system. The application program uses only GLUT function and can be recompiled with the GLUT library for other window system. OpenGL makes a heavy use of Macros to increase code readability and avoid the use of magic numbers. In most implementation, one of the include lines.

**Chapter 3**

**ABOUT THE PROJECT**

**3.1 Overview**

This project is on “3D Classroom” Computer Graphics using OpenGL Functions. It is a user interactive program where the user can view the required display by making use of the input devices like Keyboard and Mouse. This project mainly consists of a Classroom of CSE Department with all the components such as chairs, whiteboard, fan, window, etc. The classroom can be viewed in any direction using mouse and the movement can be done with the help of the keyboard.

**3.2 Objective**

* The aim of the project is to demonstrate the 3D Classroom in 3D with multiple views.
* As Linux doesn’t provide graphics editor, it should be designed in such a way that it provides a very useful graph implementation interface.
* It should be easy to understand, user interactive interface.
* Creation of primitives, such as polygons.
* Providing human interaction through mouse and keyboard.

**3.3 User Interface**

A set of keys are used to change the following:

* User can select the view of the seminar hall using mouse and keyboard.
* Zooming in is controlled by the UP arrow key.
* Zooming out is controlled by the DOWN arrow key.
* To move the camera towards left, key to be used is LEFT arrow key.
* To move the camera towards right, key to be used is RIGHT arrow key.
* To reset the scene, key to be used is R.

**3.4 Built-in Functions**

* glutInit() : interaction between the windowing system and OPENGL is initiated.
* glutInitDisplayMode() : used when double buffering is required and depth information is required.
* glutCreateWindow() : this opens the OPENGL window and displays the title at top of the window.
* glutInitWindowSize() : specifies the size of the window.
* glutInitWindowPosition() : specifies the position of the window in screen co-ordinates.
* glutKeyboardFunc() : handles normal ascii symbols.
* glutSpecialFunc() : handles special keyboard keys.
* glutReshapeFunc() : sets up the callback function for reshaping the window.
* glutIdleFunc() : this handles the processing of the b5ackground.
* glutDisplayFunc() : this handles redrawing of the window.
* glutMainLoop() : this starts the main loop, it never returns.
* glViewport() : used to set up the viewport.
* glVertex3fv() : used to set up the points or vertices in three dimensions
* glColor3fv() : used to render color to faces.
* glFlush() : used to flush the pipeline.
* glutPostRedisplay() : used to trigger an automatic redrawal of the object.
* glMatrixMode() : used to set up the required mode of the matrix.
* glLoadIdentity() : used to load or initialize to the identity matrix.
* glTranslatef() : used to translate or move the rotation centre from one point to another in three dimensions.
* glRotatef() : used to rotate an object through a specified rotation angle.

**Chapter 4**

**IMPLEMENTATION**

**4.1 Source Code**

#include <windows.h>

#include<GL/gl.h>

#include <GL/glu.h>

#include <GL/glut.h>

#include <stdlib.h>

#include <stdio.h>

#include<math.h>

#include<bits/stdc++.h>

using namespace std;

const int width = 700;

const int height = 700;

GLfloat eyeX = 31;

GLfloat eyeY = 5;

GLfloat eyeZ = -27;

GLfloat centerX = 7;

GLfloat centerY = 5;

GLfloat centerZ = 28;

double angle = 0;

bool l\_on1 = true;

bool l\_on2 = true;

bool l\_on3 = true;

bool l\_on4 = false;

float rot = -12;

int stop = 1;

float door\_angle = .5;

float l\_height = .5;

float spt\_cutoff = 30;

unsigned int ID;

vector<int>v;

static void getNormal3p(GLfloat x1, GLfloat y1, GLfloat z1, GLfloat x2, GLfloat y2, GLfloat z2, GLfloat x3, GLfloat y3, GLfloat z3)

{

GLfloat Ux, Uy, Uz, Vx, Vy, Vz, Nx, Ny, Nz;

Ux = x2 - x1;

Uy = y2 - y1;

Uz = z2 - z1;

Vx = x3 - x1;

Vy = y3 - y1;

Vz = z3 - z1;

Nx = Uy \* Vz - Uz \* Vy;

Ny = Uz \* Vx - Ux \* Vz;

Nz = Ux \* Vy - Uy \* Vx;

glNormal3f(Nx, Ny, Nz);

}

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}

};

static GLubyte c\_ind[6][4] =

{

{3,1,5,7}, //front

{6,4,0,2}, //back

{2,3,7,6}, //top

{1,0,4,5}, //bottom

{7,5,4,6}, //right

{2,0,1,3} //left

};

void cube(float R = 0.5, float G = 0.5, float B = 0.5, int type = 0, float val = 1)

{

GLfloat m\_no[] = { 0, 0, 0, 1.0 };

GLfloat m\_amb[] = { R,G,B,1 };

GLfloat m\_diff[] = { R,G,B,1 };

GLfloat m\_spec[] = { 1,1,1,1 };

GLfloat m\_sh[] = { 30 };

GLfloat m\_em[] = { 1,1,1,1 };

glMaterialfv(GL\_FRONT, GL\_AMBIENT, m\_amb);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, m\_diff);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, m\_spec);

glMaterialfv(GL\_FRONT, GL\_SHININESS, m\_sh);

if (type == 1)

{

if (l\_on1)

glMaterialfv(GL\_FRONT, GL\_EMISSION, m\_em);

else

glMaterialfv(GL\_FRONT, GL\_EMISSION, m\_no);

}

else if (type == 2)

{

if (l\_on3)

glMaterialfv(GL\_FRONT, GL\_EMISSION, m\_em);

else

glMaterialfv(GL\_FRONT, GL\_EMISSION, m\_no);

}

else

glMaterialfv(GL\_FRONT, GL\_EMISSION, m\_no);

glBegin(GL\_QUADS);

for (GLint i = 0; i < 6; i++)

{

getNormal3p(v\_cube[c\_ind[i][0]][0], v\_cube[c\_ind[i][0]][1], v\_cube[c\_ind[i][0]][2],

v\_cube[c\_ind[i][1]][0], v\_cube[c\_ind[i][1]][1], v\_cube[c\_ind[i][1]][2],

v\_cube[c\_ind[i][2]][0], v\_cube[c\_ind[i][2]][1], v\_cube[c\_ind[i][2]][2]);

glTexCoord2f(0, val);

glVertex3fv(&v\_cube[c\_ind[i][0]][0]);

glTexCoord2f(0, 0);

glVertex3fv(&v\_cube[c\_ind[i][1]][0]);

glTexCoord2f(val, 0);

glVertex3fv(&v\_cube[c\_ind[i][2]][0]);

glTexCoord2f(val, val);

glVertex3fv(&v\_cube[c\_ind[i][3]][0]);

}

glEnd();

}

class BmpLoader

{

public:

unsigned char\* textureData;

int iWidth, iHeight;

BmpLoader(const char\*);

~BmpLoader();

private:

BITMAPFILEHEADER bfh;

BITMAPINFOHEADER bih;

};

BmpLoader::BmpLoader(const char\* filename)

{

FILE\* file = 0;

file = fopen(filename, "rb");

if (!file)

cout << "File not found" << endl;

fread(&bfh, sizeof(BITMAPFILEHEADER), 1, file);

if (bfh.bfType != 0x4D42)

cout << "Not a valid bitmap" << endl;

fread(&bih, sizeof(BITMAPINFOHEADER), 1, file);

if (bih.biSizeImage == 0)

bih.biSizeImage = bih.biHeight \* bih.biWidth \* 3;

textureData = new unsigned char[bih.biSizeImage];

fseek(file, bfh.bfOffBits, SEEK\_SET);

fread(textureData, 1, bih.biSizeImage, file);

unsigned char temp;

for (int i = 0; i < bih.biSizeImage; i += 3)

{

temp = textureData[i];

textureData[i] = textureData[i + 2];

textureData[i + 2] = temp;

}

iWidth = bih.biWidth;

iHeight = bih.biHeight;

fclose(file);

}

BmpLoader::~BmpLoader()

{

delete[] textureData;

}

void LoadTexture(const char\* filename)

{

glGenTextures(1, &ID);

glBindTexture(GL\_TEXTURE\_2D, ID);

glPixelStorei(GL\_UNPACK\_ALIGNMENT, ID);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

BmpLoader bl(filename);

gluBuild2DMipmaps(GL\_TEXTURE\_2D, GL\_RGB, bl.iWidth, bl.iHeight, GL\_RGB, GL\_UNSIGNED\_BYTE, bl.textureData);

}

static void res(int width, int height)

{

glViewport(0, 0, width, height);

}

void axes()

{

float length = 10;

float width = 0.3;

// X-axis

glPushMatrix();

glTranslatef(length / 2, 0, 0);

glScalef(length, width, width);

glTranslatef(-0.5, -0.5, -0.5);

cube(0.8, 0.1, 0.1);

glPopMatrix();

// Y-axis

glPushMatrix();

glTranslatef(0, length / 2, 0);

glScalef(width, length, width);

glTranslatef(-0.5, -0.5, -0.5);

cube(0.1, 0.8, 0.1);

glPopMatrix();

// Z-axis

glPushMatrix();

glTranslatef(0, 0, length / 2);

glScalef(width, width, length);

glTranslatef(-0.5, -0.5, -0.5);

cube(0.1, 0.1, 0.8);

glPopMatrix();

}

void cse()

{

glPushMatrix();

glScalef(.3, .5, 1);

glTranslatef(45, 15, 1);

glPushMatrix();

//C

glScalef(.4, 1.5, 2.9);

glTranslatef(45, 3, 2);

float length = 3.4;

float width = .4;

//ground

float a = 1, b = 0, c = .5;

glPushMatrix();

glTranslatef(length / 2, 1, 1);;

glScalef(3 \* length, width, width);

glTranslatef(-.75, -0.5, -0.5);

cube(a, b, c);

glPopMatrix();

//right

glPushMatrix();

glPushMatrix();

glTranslatef(1, length / 2, 1);;

glScalef(width + 3, length + .5, width);

glTranslatef(0, -0.2, -0.5);

cube(a, b, c);

glPopMatrix();

glPushMatrix();

glTranslatef(length / 2, 1, 1);;

glScalef(length \* 3, width, width);

glTranslatef(-.75, 8.7, -0.5);

cube(a, b, c);

glPopMatrix();

glPopMatrix();

glPopMatrix();

//E

glPushMatrix();

glScalef(.4, 1.5, 2.9);

glTranslatef(15, 3, 2);

//ground

glPushMatrix();

glTranslatef(length / 2, 1, 1);;

glScalef(3 \* length, width, width);

glTranslatef(-.75, -0.5, -0.5);

cube(a, b, c);

glPopMatrix();

//right

glPushMatrix();

glPushMatrix();

glTranslatef(1, length / 2 + .1, 1);;

glScalef(width + 3, length, width);

glTranslatef(0, -0.2, -0.5);

cube(a, b, c);

glPopMatrix();

//top

glPushMatrix();

glTranslatef(length / 2, 1, 1);;

glScalef(length \* 3, width, width);

glTranslatef(-.75, 8.7, -0.5);

cube(a, b, c);

glPopMatrix();

//right

/\*

\*/

//rack4

glPushMatrix();

glTranslatef(length, 1, 1);;

glScalef(3 \* length, width, width);

glTranslatef(-.94, 4, -0.5);

cube(a, b, c);

glPopMatrix();

glPopMatrix();

glPopMatrix();

glPushMatrix();

glScalef(.4, 1.5, 2.9);

glTranslatef(30, 3, 2);

//ground

glPushMatrix();

glTranslatef(length / 2, 1, 1);;

glScalef(3 \* length, width, width);

glTranslatef(-.75, -0.5, -0.5);

cube(a, b, c);

glPopMatrix();

//right

glPushMatrix();

glPushMatrix();

glTranslatef(1, length / 2, 1);;

glScalef(width + 3, length / 2, width);

glTranslatef(0, 0.7, -0.5);

cube(a, b, c);

glPopMatrix();

glPushMatrix();

glTranslatef(1, length / 2, 1);;

glScalef(width + 3, length / 2 + .1, width);

glTranslatef(-2, -0.5, -0.5);

cube(a, b, c);

glPopMatrix();

//top

glPushMatrix();

glTranslatef(length / 2, 1, 1);;

glScalef(length \* 3, width, width);

glTranslatef(-.75, 8.7, -0.5);

cube(a, b, c);

glPopMatrix();

//rack4

glPushMatrix();

glTranslatef(length, 1, 1);;

glScalef(3 \* length, width, width);

glTranslatef(-.9, 4, -0.5);

cube(a, b, c);

glPopMatrix();

glPopMatrix();

glPopMatrix();

glPopMatrix();

}

void flr()

{

for (int i = -10; i <= 35; i++)

{

for (float j = -15; j <= 14; j += 2)

{

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(2, .5, 1);

glTranslatef(j, -.999, -i);

if (i % 2 == 0)

cube(1, 1, 1);

else

cube(0, 0, .5);

glPopMatrix();

}

}

for (int i = -10; i <= 35; i++)

{

for (float j = -14; j <= 14; j += 2)

{

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(2, .5, 1);

glTranslatef(j, -.999, -i);

if (i % 2 != 0)

cube(1, 1, 1);

else

cube(0, 0, .5);

glPopMatrix();

}

}

}

void wall1()

{

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, v[0]);

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(1, 20, 60);

glTranslatef(-30, 0, -0.5);

cube(1, 1, 1, 0, 2);

glPopMatrix();

glDisable(GL\_TEXTURE\_2D);

}

void wall2()

{

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, v[0]);

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(1, 20, 60);

glTranslatef(29, 0, -0.5);

cube(1, 1, 1, 0, 2);

glPopMatrix();

glDisable(GL\_TEXTURE\_2D);

}

void wall3()

{

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, v[0]);

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(60, 20, 1);

glTranslatef(-.5, 0, 10);

cube(0.741, 0.718, 0.420, 0, 2);

glPopMatrix();

glDisable(GL\_TEXTURE\_2D);

}

void door()

{

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(50, 20, 1);

glTranslatef(-.6, 0, -30);

cube(1, .69, .0);

glPopMatrix();

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(42, 10, 1);

glTranslatef(-.3, 1, -30);

cube(1, .69, .0);

glPopMatrix();

}

void board()

{

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(30, 10, 1);

glTranslatef(-.6, .4, 9);

cube(1, 1, 1);

glPopMatrix();

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(31.5, 11, 1);

glTranslatef(-.601, .32, 9.5);

cube(0, 0, 0);

glPopMatrix();

}

void window()

{

glPushMatrix();

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(1, 8, 15);

glTranslatef(28, .5, -1);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, v[1]);

cube(1, 1, 1, 0, 1);

glBindTexture(GL\_TEXTURE\_2D, v[0]);

glPopMatrix();

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(1, 8, .2);

glTranslatef(28, .5, -76);

cube(1, .59, .0);

glDisable(GL\_TEXTURE\_2D);

glPopMatrix();

glRotatef(door\_angle, 0, 1, 0);

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(1, .2, 15);

glTranslatef(27.9, 50, -1);

cube(0, 0, 0);

glPopMatrix();

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(1, .2, 15);

glTranslatef(27.9, 30, -1);

cube(0, 0, 0);

glPopMatrix();

for (int i = 0; i >= -75; i -= 2)

{

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(1, 8, .2);

glTranslatef(27.9, .5, i);

cube(0, 0, 0);

glPopMatrix();

}

glPopMatrix();

}

void headwall()

{

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, v[2]);

glPushMatrix();

// glTranslatef(0,-0.5,0);

glScalef(60, 1, 60);

glTranslatef(-0.5, 19, -0.5);

cube(0.690, 0.769, 0.871, 0, 4);

glPopMatrix();

glDisable(GL\_TEXTURE\_2D);

}

float k = 0;

float angle1 = 0;

bool markdoor = 0;

double window\_val = 0;

void spin()

{

angle = (angle + k) \* stop;

k += .001;

if (angle >= 360)

angle = 0;

angle1 += .2;

if (angle1 >= 360)

angle1 = 0;

bool ok = 0;

bool ok1 = 0;

if (markdoor && door\_angle > -.8)

{

door\_angle -= .001, ok = 1;

if (door\_angle < .4)

window\_val += .00085;

window\_val = min(window\_val, 1.00);

l\_on4 = 1;

}

if (!markdoor && door\_angle < .5)

{

door\_angle += .001, ok = 1;

window\_val -= .001;

window\_val = max(window\_val, 0.00);

}

if (!ok)

l\_on4 = markdoor;

}

double a = 2;

double b = 1;

double c = 10;

void fan()

{

//fan stand

float length = 10;

float width = 0.3;

glPushMatrix();

glTranslatef(1, length / 2, -4);

glScalef(width, length, width);

glTranslatef(-20, 1, 20);

cube(0.4, 0.4, 0.1);

glPopMatrix();

glPushMatrix();

glScalef(1, .1, 1);

glTranslatef(-5, 150, 2);

cube(1, 1, 1); // Set color as glColor3f(R,G,B)

glRotatef(angle, 0, 0, 1);

glRecti(-a, -a, a, a);

cube(1, 0, 0);

glRecti(-b, a, b, c);

glRecti(-c, -b, -a, b);

glRecti(-b, -c, b, -a);

glRecti(a, -b, c, b);

glPopMatrix();

}

void Chair12()

{

float height = 2;

float width = 4;

float length = 2;

float base\_height = .5;

float leg\_height = height - base\_height;

float leg\_width = .4;

// whole table

glPushMatrix();

glTranslatef(0, leg\_height, 0);

// base

glPushMatrix();

glScalef(width, base\_height, length);

glTranslatef(-0.5, 0, -0.5);

cube(0.4f, 0.302f, 0.0f);

glPopMatrix();

// legs

glPushMatrix();

glTranslatef((width / 2 - leg\_width / 2), 0, (length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(0.4f, 0.302f, 0.0f);

glPopMatrix();

glPushMatrix();

glTranslatef((width / 2 - leg\_width / 2), 0, -(length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(0.4f, 0.302f, 0.0f);

glPopMatrix();

glPushMatrix();

glTranslatef(-(width / 2 - leg\_width / 2), 0, (length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(0.4f, 0.302f, 0.0f);

glPopMatrix();

glPushMatrix();

glTranslatef(-(width / 2 - leg\_width / 2), 0, -(length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(0.4f, 0.302f, 0.0f);

glPopMatrix();

//upper put

glPushMatrix();

glScalef(3, 2, .5);

glTranslatef(-0.5, .5, -1);

cube(1, 1, 0);

glPopMatrix();

//middle part 1

glPushMatrix();

glScalef(.2, 1, .2);

glTranslatef(3, 0, -2);

cube(0, 0, 0);

glPopMatrix();

glPushMatrix();

glScalef(.2, 1, .2);

glTranslatef(-4, 0, -2);

cube(0, 0, 0);

glPopMatrix();

glPopMatrix();

}

void Table12(float height = 2, float width = 4, float length = 2, float base\_height = .5, float leg\_width = .2)

{

float leg\_height = height - base\_height;

// whole table

glPushMatrix();

glTranslatef(0, leg\_height, 0);

// base

glPushMatrix();

glScalef(width, base\_height, length);

glTranslatef(-0.5, 0, -0.5);

cube(.6, .2, 0);

glPopMatrix();

// legs

glPushMatrix();

glTranslatef((width / 2 - leg\_width / 2), 0, (length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(.8f, .6f, 0.0f);

glPopMatrix();

glPushMatrix();

glTranslatef((width / 2 - leg\_width / 2), 0, -(length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(.8f, .6f, 0.0f);

glPopMatrix();

glPushMatrix();

glTranslatef(-(width / 2 - leg\_width / 2), 0, (length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(.8f, .6f, 0.0f);

glPopMatrix();

glPushMatrix();

glTranslatef(-(width / 2 - leg\_width / 2), 0, -(length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(.8f, .6f, 0.0f);

glPopMatrix();

glPopMatrix();

}

void test()

{

//teacher chair

glPushMatrix();

glTranslatef(10.0, 1, -14 + 9 \* 2.0 + 2.2f);

glScalef(0.8f, 0.8f, 0.8f);

glRotatef(180.0, 0.0, 1.0, 0.0);

Chair12();

glPopMatrix();

//teacher table

glPushMatrix();

glTranslatef(10.0, 1, -16 + 9 \* 2.0 + 2.2f);

glScalef(1.2f, 1.3f, 0.8f);

Table12(2, 4, 2, .6, .3);

glPopMatrix();

for (int i = -8; i <= 10; i += 2)

{

for (int j = -4; j <= 6; j += 2)

{

glPushMatrix();

glTranslatef(i \* 2.0, 0.6f, -16 + j \* 2.0 + 2.2f);

glScalef(0.5f, 0.5f, 0.5f);

glRotatef(0.0, 0.0, 1.0, 0.0);

Chair12();

glPopMatrix();

}

}

for (int i = -8; i <= 10; i += 2)

{

for (int j = -4; j <= 6; j += 2)

{

glPushMatrix();

glTranslatef(i \* 2.0 + 0.3f, 1.2f, -14.2 + j \* 2.0 + 1.2f);

glScalef(0.75f, 0.7f, 0.6f);

Table12();

glPopMatrix();

}

}

}

void table()

{

float height = 1;

float width = 30;

float length = 5;

float base\_height = .5;

float leg\_height = height - base\_height;

float leg\_width = 1;

// whole table

glPushMatrix();

glTranslatef(0, 0, 5);

glPushMatrix();

glTranslatef(0, leg\_height, 0);

// base

glPushMatrix();

glScalef(width, base\_height, length);

glTranslatef(-0.5, 0, -0.5);

cube(0.53, 0.39, 0.28);

glPopMatrix();

// legs

glPushMatrix();

glTranslatef((width / 2 - leg\_width / 2), 0, (length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(0.53, 0.39, 0.28);

glPopMatrix();

glPushMatrix();

glTranslatef((width / 2 - leg\_width / 2), 0, -(length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(0.53, 0.39, 0.28);

glPopMatrix();

glPushMatrix();

glTranslatef(-(width / 2 - leg\_width / 2), 0, (length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(0.53, 0.39, 0.28);

glPopMatrix();

glPushMatrix();

glTranslatef(-(width / 2 - leg\_width / 2), 0, -(length / 2 - leg\_width / 2));

glScalef(leg\_width, leg\_height, leg\_width);

glTranslatef(-0.5, -1, -0.5);

cube(0.53, 0.39, 0.28);

glPopMatrix();

glPopMatrix();

glPopMatrix();

}

void drop()

{

glPushMatrix();

//wear drop front

cube(113 / 255.0, 74 / 255.0, 44 / 255.0);

//glColor3ub (113,74,44);

glBegin(GL\_POLYGON);

glVertex3f(460, 660, 0);

glVertex3f(470, 670, 0);

glVertex3f(330, 670, 0);

glVertex3f(340, 660, 0);

glVertex3f(340, 320, 0);

glVertex3f(460, 320, 0);

glEnd();

//wear drop front-drawer 3 - outer

glPopMatrix();

glPushMatrix();

glTranslatef(0, 0, 0);

glScalef(105, 95, 10);

glPushMatrix();

glTranslatef(3.32, 4.6, -.49);

cube(62 / 255.0, 43 / 255.0, 45 / 255.0);

glPopMatrix();

glPushMatrix();

glTranslatef(3.32, 5.7, -.49);

cube(62 / 255.0, 43 / 255.0, 45 / 255.0);

glPopMatrix();

glPushMatrix();

glTranslatef(3.32, 3.5, -.49);

cube(62 / 255.0, 43 / 255.0, 45 / 255.0);

glPopMatrix();

glPopMatrix();

//wear drop front-drawer 1 - handel

glPushMatrix();

glTranslatef(1, 0, 0);

glScalef(20, 10, 10);

glPushMatrix();

glTranslatef(19.5, 38, -.5);

cube(1, 1, 1);

glPopMatrix();

glPushMatrix();

glTranslatef(19.5, 58, -.5);

cube(1, 1, 1);

glPopMatrix();

glPushMatrix();

glTranslatef(19.5, 48, -.5);

cube(1, 1, 1);

glPopMatrix();

glPopMatrix();

//wear drop side

cube(62 / 255.0, 43 / 255.0, 35 / 255.0);

glColor3ub(62, 43, 35);

glBegin(GL\_POLYGON);

glVertex3f(470, 670, 0);

glVertex3f(500, 667, 0);

glVertex3f(492, 657, 0);

glVertex3f(492, 330, 0);

glVertex3f(460, 320, 0);

glVertex3f(460, 660, 0);

glEnd();

glPopMatrix();

}

void weardrop()

{

glPushMatrix();

// glTranslatef(0,-0.5,0);

glTranslatef(9, -7, 8);

//glRotatef(rot,0,1,1);

glScalef(.04, .022, .03);

drop();

glPopMatrix();

}

void clock1()

{

glPushMatrix();

//clock outer

glPushMatrix();

glScalef(1.5, 1.3, 1);

glTranslatef(-220, -200, 1);

cube(0, 0, 0);

glBegin(GL\_QUADS);

glVertex3f(680, 800, 0);

glVertex3f(630, 800, 0);

glVertex3f(630, 900, 0);

glVertex3f(680, 900, 0);

glEnd();

//clock inner

glPushMatrix();

glTranslatef(0, 0, -.98);

cube(0.847, 0.749, 0.847);

glBegin(GL\_QUADS);

glVertex3f(675, 805, 0);

glVertex3f(635, 805, 0);

glVertex3f(635, 895, 0);

glVertex3f(675, 895, 0);

glEnd();

glPopMatrix();

glPopMatrix();

glPushMatrix();

glTranslatef(0, 0, -2);

////////////////////////////

glPushMatrix();

glTranslatef(653, 847, 1);

glRotatef(angle1, 0, 0, 1);

cube(0, 0, 0);

glRecti(-b, a, b, 3 \* c);

glPopMatrix();

/////////////////////////////

//clock second

cube(0, 0, 0);

glBegin(GL\_LINES);

glVertex3f(655, 850, 0);

glVertex3f(655, 870, 0);

glEnd();

//clock hour

cube(0, 0, 0);

glBegin(GL\_LINES);

//glBegin(GL\_POINT\_SIZE);

glVertex3f(655, 850, 0);

glVertex3f(645, 810, 0);

glEnd();

glPopMatrix();

glPopMatrix();

}

void clock2()

{

glPushMatrix();

// glTranslatef(0,-0.5,0);

glTranslatef(-75, -30, 9);

//glRotatef(rot,0,1,1);

glScalef(.08, .05, .05);

clock1();

glPopMatrix();

}

void fan1()

{

glPushMatrix();

glTranslatef(1, -.5, -15);

fan();

glPopMatrix();

}

float al1, dl1, sl1;

float al2, dl2, sl2;

float al3, dl3, sl3;

void light1(float a, float b, float c)

{

glEnable(GL\_LIGHT0);

//light

GLfloat l\_no[] = { 0, 0, 0, 1.0 };

GLfloat l\_amb[] = { 0.4 + al1, 0.4 + al1, 0.4 + al1, 1.0 };

GLfloat l\_dif[] = { .4 + dl1,.4 + dl1,.4 + dl1,1 };

GLfloat l\_spec[] = { .4 + sl1,.4 + sl1,.4 + sl1,1 };

GLfloat l\_pos[] = { a,b,c,1.0 };

if (l\_on1)

glLightfv(GL\_LIGHT0, GL\_AMBIENT, l\_amb);

else

glLightfv(GL\_LIGHT0, GL\_AMBIENT, l\_no);

if (l\_on1)

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, l\_dif);

else

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, l\_no);

if (l\_on1)

glLightfv(GL\_LIGHT0, GL\_SPECULAR, l\_spec);

else

glLightfv(GL\_LIGHT0, GL\_SPECULAR, l\_no);

glLightfv(GL\_LIGHT0, GL\_POSITION, l\_pos);

}

void light2(float a, float b, float c)

{

glEnable(GL\_LIGHT2);

//light

GLfloat l\_no[] = { 0, 0, 0, 1.0 };

GLfloat l\_amb[] = { 0.3 + al2, 0.3 + al2, 0.3 + al2, 1.0 };

GLfloat l\_dif[] = { .6 + dl2,.6 + dl2,.6 + dl2,1 };

GLfloat l\_spec[] = { 5 + sl2,5 + sl2,5 + sl2,1 };

GLfloat l\_pos[] = { a,b,c,1.0 };

if (l\_on3)

glLightfv(GL\_LIGHT2, GL\_AMBIENT, l\_amb);

else

glLightfv(GL\_LIGHT2, GL\_AMBIENT, l\_no);

if (l\_on3)

glLightfv(GL\_LIGHT2, GL\_DIFFUSE, l\_dif);

else

glLightfv(GL\_LIGHT2, GL\_DIFFUSE, l\_no);

if (l\_on3)

glLightfv(GL\_LIGHT2, GL\_SPECULAR, l\_spec);

else

glLightfv(GL\_LIGHT2, GL\_SPECULAR, l\_no);

glLightfv(GL\_LIGHT2, GL\_POSITION, l\_pos);

// GLfloat l\_spt[] = {0,-1,0,1};

// GLfloat spt\_ct[] = {66};

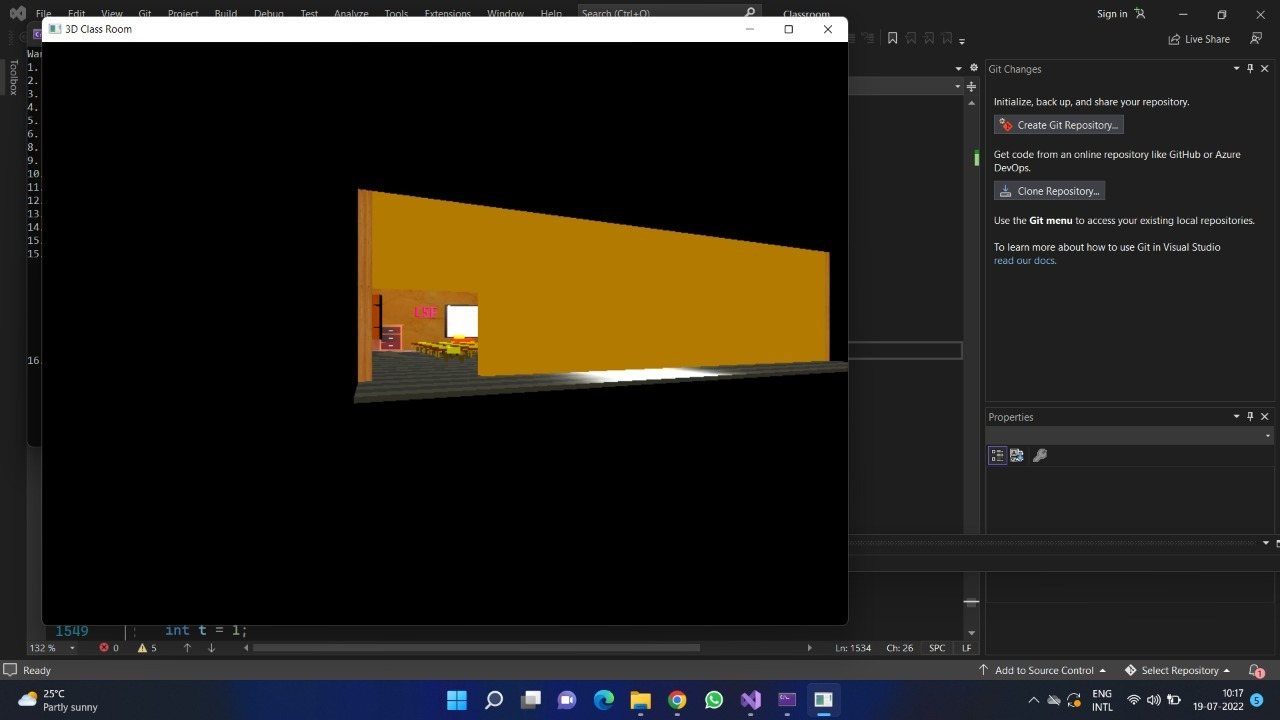
// glLightfv(GL\_LIGHT2, GL\_SPOT\_DIRECTION, l\_spt);

// glLightfv(GL\_LIGHT2, GL\_SPOT\_CUTOFF, spt\_ct);

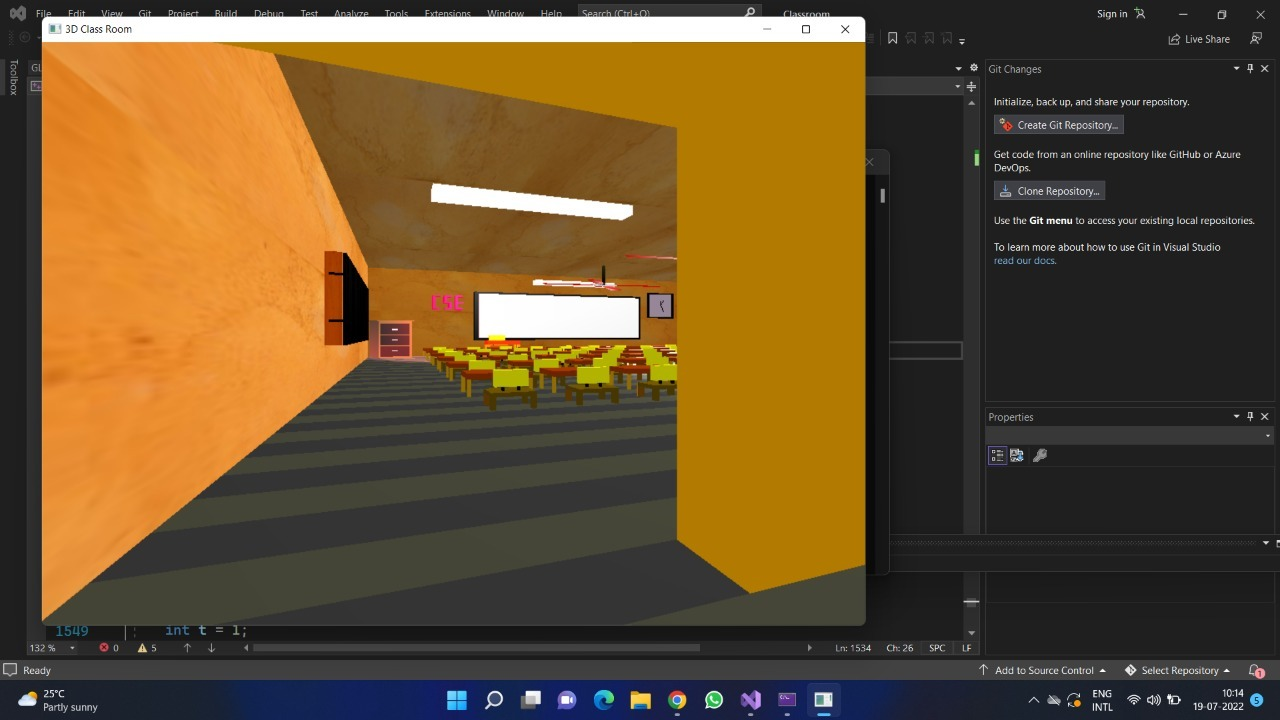
}

**Chapter 5**

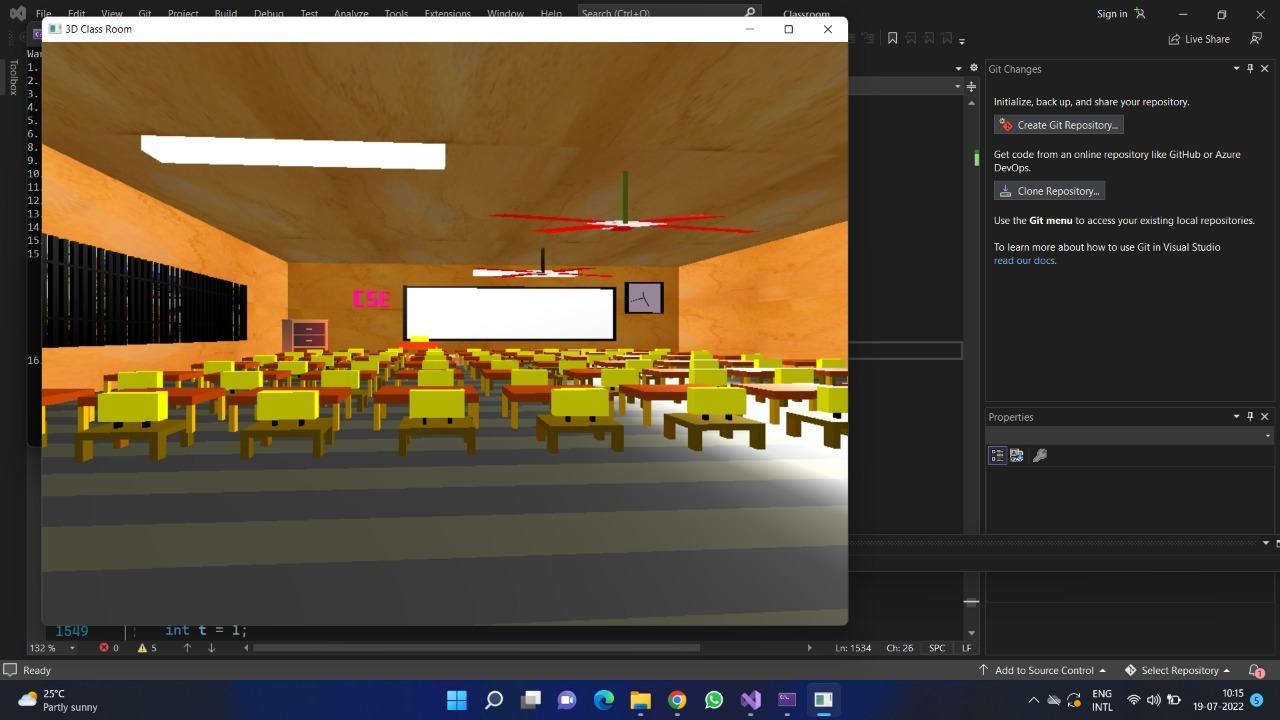
**RESULTS**



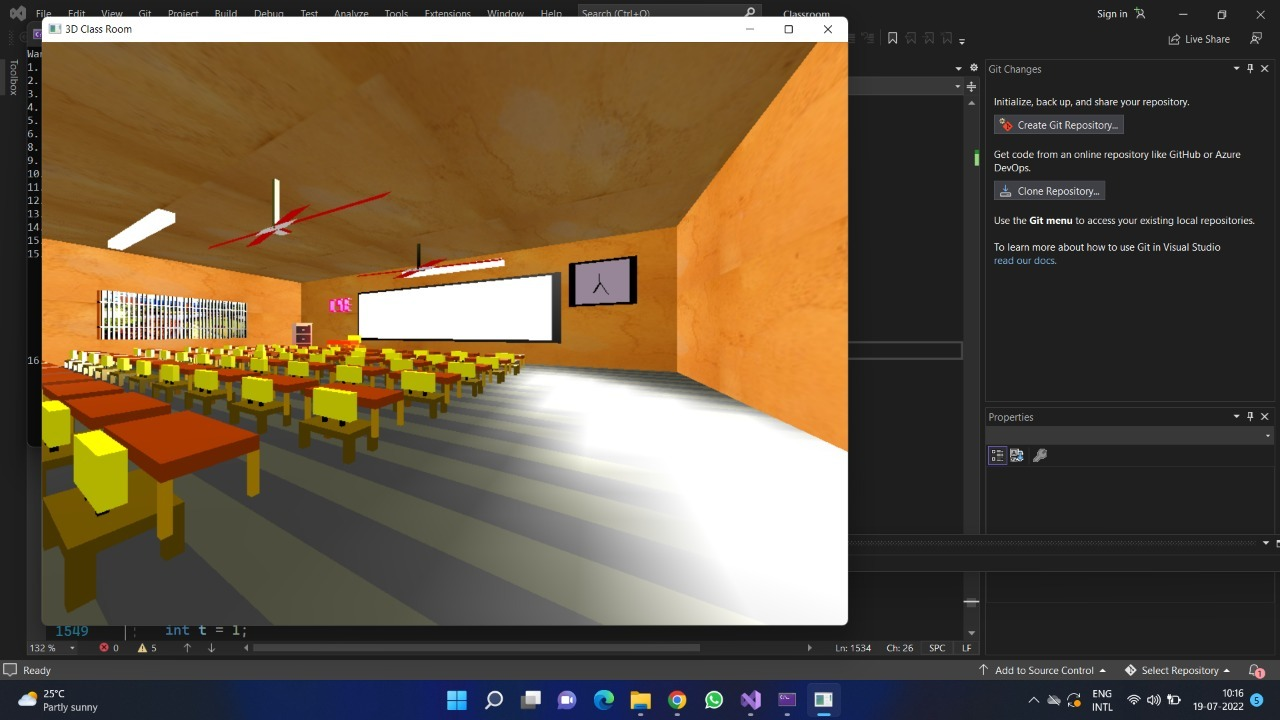
**Fig 5.1: Cube view**



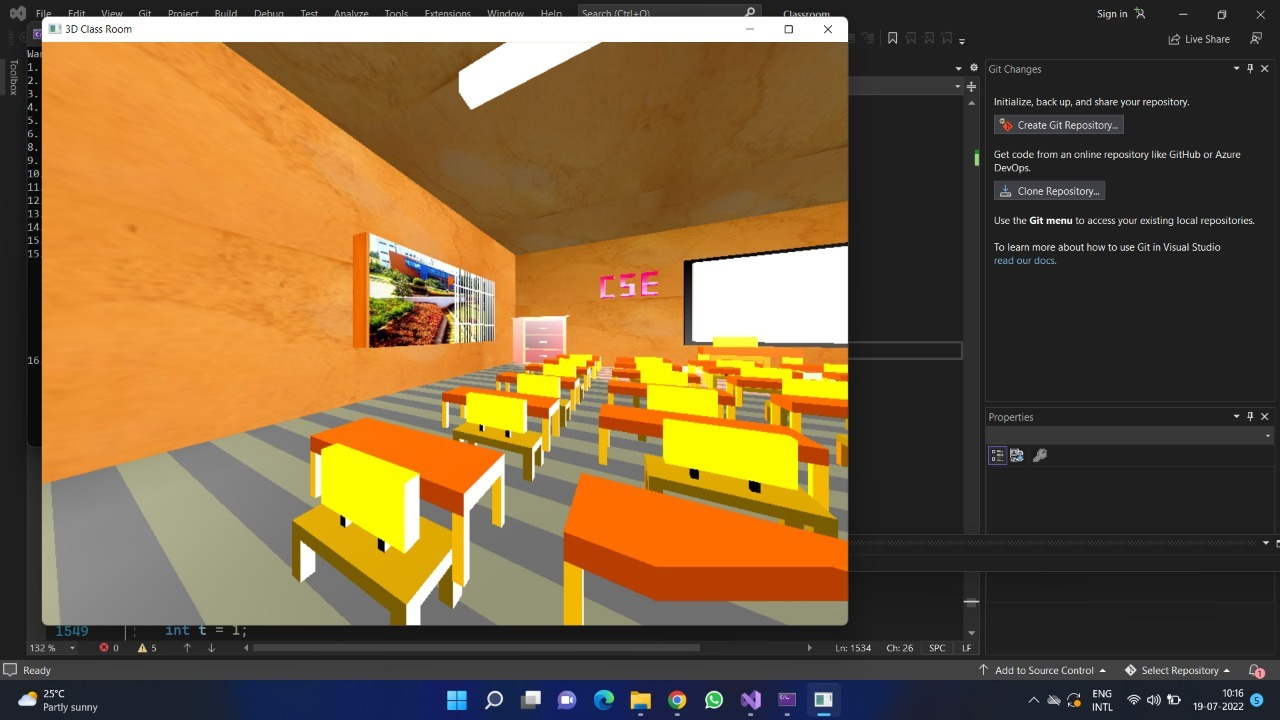
**Fig 5.2: Door view**



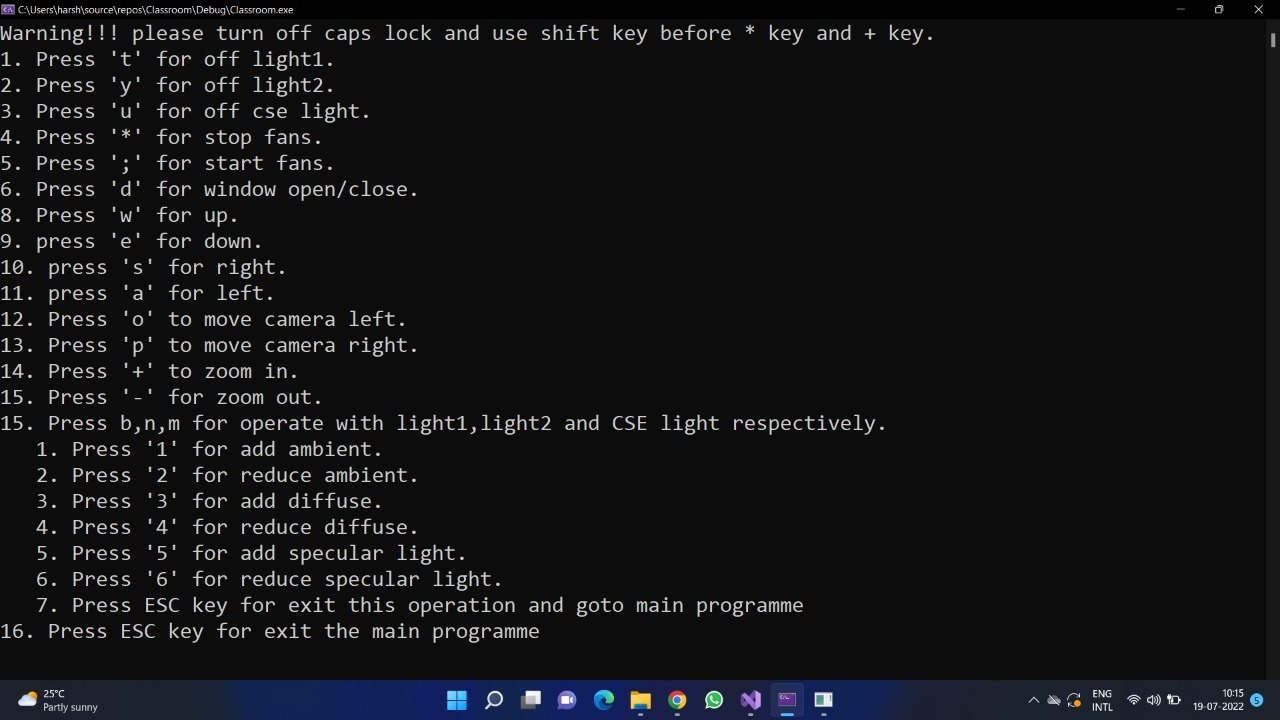
**Fig 5.3 : Classroom front view**

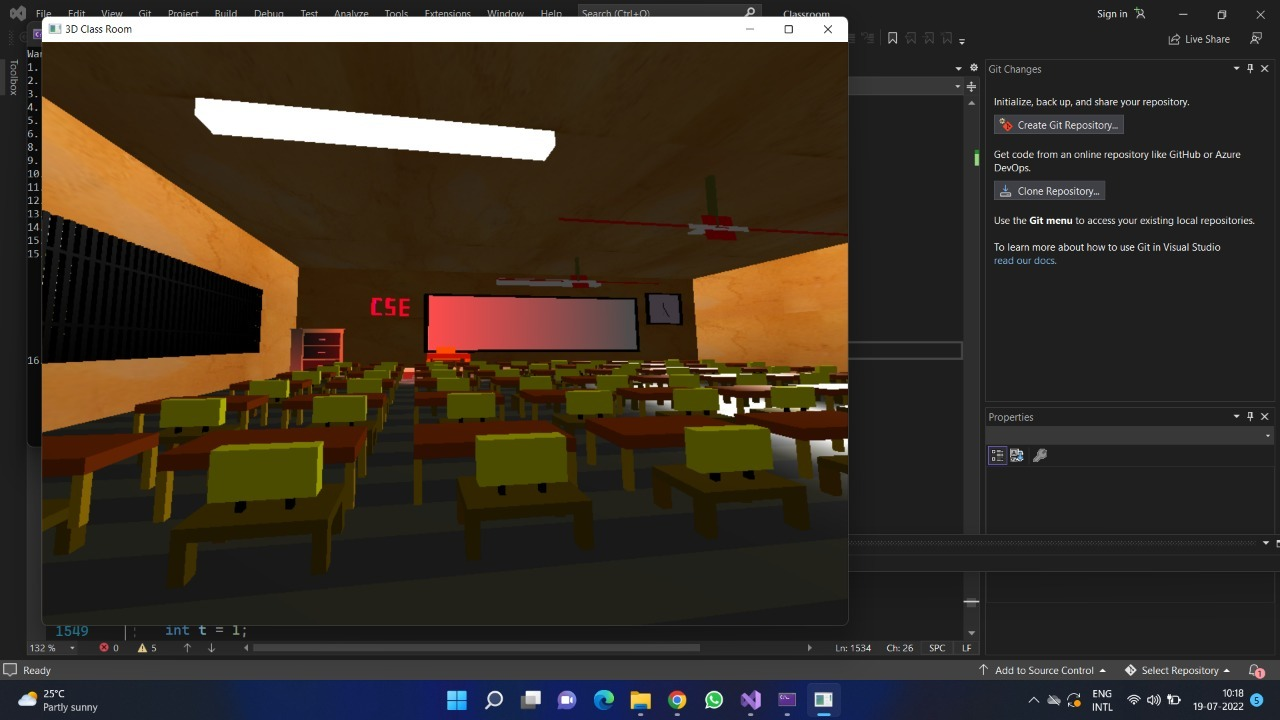


**Fig 5.4 : Classroom side view**

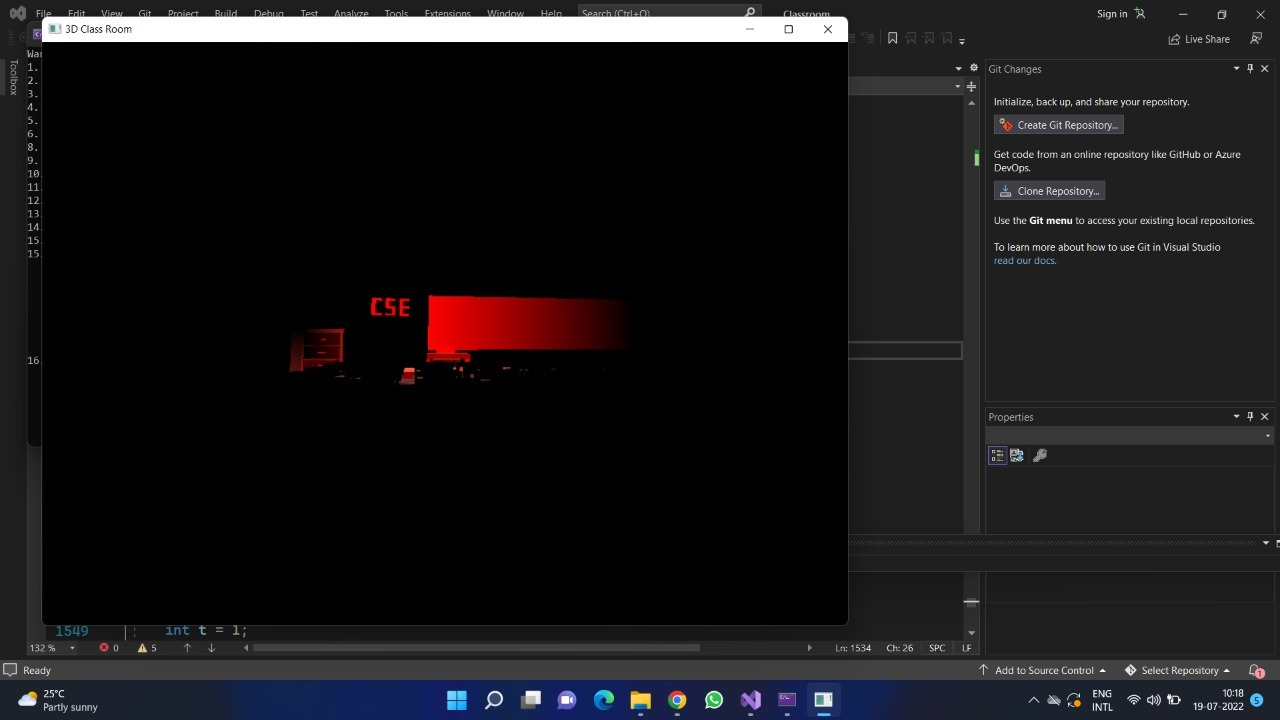


**Fig 5.5 : Window view**

 **Fig 5.6 : Keyboard Function to use trace the output with having the option frontview window view and light function**



**Fig 5.7 : Light 1 off , press ‘t’ Lights off 1 , the first light will trun off in the room**



**Fig 5.7 : Light 1 and 2 off ,press ‘y’ to turn on and turn off the both lights**

**Chapter 6**

**CONCLUSION AND FUTURE WORK**

**6.1 Conclusion**

The project 3D Classroom has been tested under Ubuntu **20.04.4 LTS and has been found to provide ease of use and manipulation to the user. The project created for Ubuntu 20.04.4 LTS operating system can be used to draw lines, boxes, circles, ellipses and polygons. It has a very simple and effective user interface.**

**We found designing and developing this 3D Classroom as a very interesting and learning experience. It helped us to learn about computer graphics, design of Graphical User Interfaces, interface to the user, user interaction handling and screen management. The graphics editor provides all and more than the features that have been detailed in the university syllabus.**

**6.2 Future Enhancement**

**These are the features that are planned to be supported in the future:**

* Addition of students and teachers.
* More interactive objects.
* To improve the look of the Classroom.
* To implement shadow using built-in function.

**BIBLIOGRAPHY**

**Book References**

[1] Interactive Computer Graphics – A top down approach using OpenGL—by Edward Angle.

[2] Jackie L. Neider, Mark Warhol, Tom R. Davis, “OpenGL Red Book”, Second Revised Edition, 2005.

[3] Donald D Hearn and M. Pauline Baker, “Computer Graphics with OpenGL”, 3rd Edition.

**Web References**

[1] www.opengl.org

[2] www.vtupro.blogspot.in

[3] www.opengl.start4all.com

[4] www.youtube.com

[5] https://stackoverflow.com