

MINIPROJECT

|  |  |
| --- | --- |
| **Roll No and Name of the Student** | 1.SAGAR G |
| 2.CHETHAN D N |
| 3.NIKHIL M S |
| 4.RAJAVARDHAN R |
| **Semester / Year** | 3rd SEMESTER / 2nd YEAR |
| **Department** | DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING |
| **Subject of Mini-Project** | COMPUTER GRAPHICS |
| **Mini-Project Topic** | 2D ROTATIVE SHAPE(TRIANGLE) |
| **Date of Submission** | 25/12/2022 |

**Mini-Project Mark Allocation**

|  |  |  |
| --- | --- | --- |
| **Mini-Project Mark Allocation** | **Total Mark** | **Mark Achieved** |
| **Mini-Project Structure and Presentation**  **(A)** | **5** |  |
| **Mini-Project demo(B)** | **10** |  |
| **Mini-Project Report(C)** | **5** |  |
| **Total (A+B+C)** | **20** |  |

General Comments:

Faculty Signature: Faculty Name: Date:

# INTRODUCTION

**1.1 Introduction to the graphics:**

Computer graphics is one of the most exciting and rapidly growing computer field and computer. It is also an extremely effective medium for communication between men.

The human can understand the information content of a displayed diagram or perceptive view much faster than it can understand a table of numbers.

There is a lot of development in hardware and software required to generate images, and now-a-days the cost of such hardware and software is also dropping rapidly. Due to this the interactive computer graphics is becoming available to more and more people.

Computer graphics today is largely interactive. The user controls the contents, structure and appearance of the objects and of their displayed images by using input devices, such as keyboard, mouse or touch sensitive panel on the screen. Because of the close relationship between the input devices and display, the handling of such devices is included in the study of computer graphics.

**1.2 Uses of computer graphics:**

**User interface:**

It is now a well-established fact that graphical interfaces provide an alternative and easy interaction between users and computers the built in graphics provided with user interfaces use the control items.

In industry, business government and education organization’s computer graphics is most commonly used to create 2D and 3D graphs of mathematical, physical and economic functions in the form of histograms, bars and pie charts which are very useful in decision making.

**Computer aided drafting and design:**

The computer aided drafting uses the graphics to components and systems. Electrical, mechanical and electronic devices such as automobile bodies, structure of airplane, ships, buildings.

**Simulation and animation for scientific visualization and environment:**

Use of graphics in simulation makes mathematical models and mechanical systems more realistic and easy to study. The interactive graphics supported by animation software proved their use in production of animated movies and cartoon films.

**1.3 OpenGL**

OpenGL (open graphic library) is a standard specification defining a cross language cross platform API for writing application that produces 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex 3D scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization and flight simulation. It is also used in video games, where it competes with direct 3D on Microsoft Windows Platforms, OpenGL is managed by the nonprofit technology consortium, the Khronos group Inc.

OpenGL serves two main purposes:

* To hide the complexities of interfacing with different 3D accelerators, by presenting programmer with a single. Uniform API
* To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set.

## **2D ROTATING TRIANGLE**

Installation of CodeBlocks in Windows or freeglut in Ubuntu Operating System and demonstration of the project. After purchase, you need to send a request for Installation and Setup to vtupulse@gmail.com mentioning your Order ID and contact details. The project idea is to display different 2D objects. This graphics package is based on the OpenGL library functions. The programming language used here is C using OpenGL libraries.

In this project, we designed Triangle, Quad which is placed in a specific position in a plane, with specific colors, and for those objects we implement rotation.

The operation implemented are:

* Coloring.
* Rotation.
* Size.
* It also illustrates how to add menus.
* glBegin() and glEnd() functions can be used to draw the 2
* D objects.
* Myreshape function is called whenever the user reshapes the openGL window.

# LITERATURE SURVEY

**2.1** [**WWW.OPENGL.ORG:**](http://www.opengl.org/)

OpenGL 3.0 adds many features to the core of OpenGL. It also brings with it a deprecation model that previous versions of OpenGL did not have. Before OpenGL 3.0, anything that became core had to remain in the specification permanently. The deprecation model allows OpenGL versions to announce that certain features may be removed from the core in later versions.

The OpenGL specification now is broken into two specifications: core and compatibility. Compatibility provides full backwards compatibility with GL 2.1 and below, while Core does not. A new context creation model exists; it is the only way to create core contexts of OpenGL 3.1 and above.

Part of this new API is a specification of exactly what version of OpenGL you want. So if you ask for a GL 3.1 context, you are telling the system that you expect that any entrypoints version 3.1 removed from earlier versions will not be available, and that any entrypoints 3.1 added to new versions will be available. The new API can fail if the implementation simply does not implement that version of OpenGL.

More detailed instructions for Creating an OpenGL Context are available.

**OpenGL specifications:**

OpenGL 4.1 Core Profile Specification

OpenGL Shading Language 4.1 Specification

OpenGL 3.3 Core Profile Specification

OpenGL Shading Language 3.3 Specification

**How to make your first OpenGL Program:**

The first thing to do is chose a programming language. It could be C, C++, C#, Visual Basic, Pascal, Perl, Java, Ada, x86 assembly, etc. As long as a language has an OpenGL binding for your chosen language, you may use it.

The second thing is to choose a compiler. It could be MS Visual C++, Code::Blocks, Delphi, Masm, etc. Remember that OpenGL is an API, so as long as you have the language bindings for your compiler, you can do OpenGL programming.

Typically, a compiler comes with the binding files. For example, if you have a C++ compiler, it will come with gl.h and opengl32.lib. It may even come with glu.h and glu32.lib, glut.h and glut32.lib.

If you don't have your binding files, you will need to figure out where to download them from. Microsoft releases their Windows Platform SDK which contains these files and most likely you don't need it because your compiler came with the files.

You might want to use SDL, GLUT, freeGLUT, or some other wrapper that takes care of creating a GL window for you and destroying for you. It makes it easier for someone who just wants to learn the OpenGL API syntax.

Assuming you know how to program in your language of choice, now all you need it to learn OpenGL. There are many online tutorials. Just search for opengl+tutorial in your favorite search engine or visit some of the tutorials listed here.

**OpenGL Viewers:**

These are programs that you install and run, and they give you information specific to the OpenGL API your system implements, like the version offered by your system, the vendor, the renderer, the extension list, supported viewport size, line size, point size, plus many other details.

Some might include a benchmark. Some are standalone benchmarks.

GPU Caps Viewer (Windows XP, Vista 32)

OpenGL Extension Viewer (Windows, Windows x64 and MacOS X)

OpenGL ES benchmark tool (Linux, Symbian, Windows Mobile)

Fur rendering benchmark (Windows)

Futuremark's GL ES benchmark

**2.2 GOOGLE SEARCH:**

Google.com puts the Internet's wealth of information at your fingertips. The Google Search Appliance does the same for all your corporate information. The Google Search Appliance is an integrated hardware and software product designed to give businesses the productivity-enhancing power of Google search. It's a corporate search solution as simple, powerful and comprehensive as Google itself. The latest version adds key new features around search quality, access control and connectivity.

The Google Search Appliance makes the sea of lost and misplaced data on your web servers, file servers, content management systems, relational databases and business applications instantly available from a single familiar search box. Through an interface as simple and intuitive as Google.com, your employees will have instant, real-time secure access to all the information and knowledge across your entire enterprise – in more than 220 different file formats, and in over 109 different languages.

**HARDWARE AND SOFTWARE REQUIREMENTS**

**3.1 Hardware requirements**:

* Pentium or higher processor.
* 512 MB or more RAM
* A standard keyboard, compatible mouse and a VGA monitor

**3.2 Software requirements**:

This graphics package has been designed for UBUNTU Platform and uses ECLIPSE software

OS : Ubuntu 10.10

Development Tool : Eclipse

Language : C

# SOFTWARE DESIGN

## 4.1 Proposed System

To achieve three dimensional effects, OpenGL software is proposed. It is software which provides a graphical interface. It is an interface between application program and graphics hardware.

The advantages are:

* OpenGL is designed as a streamlined.
* It is a hardware independent interface, it can be implemented on many different hardware platforms.
* With OpenGL, we can draw a small set of geometric primitive such as points, lines and polygons etc.
* It provides double buffering which is vital in providing transformations.
* It is event driven software.
* It provides call back function.

**4.2 Detailed Design**

Transformation Functions

* Translation:

Translation is done by adding the required amount of translation quantities to each of the points of the objects in the selected area. If P(x,y) be the a point and (tx,ty) translation quantities then the translated point is given by glTranlate(dx,dy,dz);

* Rotation:

The rotation of an object by an angle ‘a’ is accomplished by rotating each of the points of the object. The rotated points can be obtained using the OpenGL functions glRotate (angle, vx,vy,vz);

* Scaling:

The scaling operation on an object can be carried out for an object by multiplying each

Of the points (x,y,z) by the scaling factors sx, sy and sz. glScale(sx,sy,sz);

# IMPLEMENTATION

**5.1 DESCRIPTION:**

GL primitives can have either flat or smooth shading. Smooth shading, the default, causes the computed colors of vertices to be interpolated as the primitive is rasterized typically assigning different colors to each resulting pixel fragment. Flat shading selects the computed color of just one vertex and assigns it to all the pixel fragments generated by rasterizing a single primitive.

**THIS PROJECT CONTAINS THE FOLLOWING KEY WORDS**

glClearColor : to screen cleared, use black. glutCreateWindow() : Tocreate window for oytput.

glVertex() : To identify the vertices it maybe int,double,float.

Glut : An introduction to the OpenGL utility ToolKit. glutCreateWindow() :Creates a top level window. glutInitWindowPosition() : set the initial window position and size.

glutInit : initial the GLUT library.

glutMainLoop() : enters the glut event processing loop.

**ALGORITHMS USED:**

#include <windows.h>

#include <gl/gl.h>

LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);

void EnableOpenGL(HWND hwnd, HDC\*, HGLRC\*);

void DisableOpenGL(HWND, HDC, HGLRC);

int WINAPI WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow)

{

WNDCLASSEX wcex;

HWND hwnd;

HDC hDC;

HGLRC hRC;

MSG msg;

BOOL bQuit = FALSE;

float theta = 0.0f;

/\* register window class \*/

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_OWNDC;

wcex.lpfnWndProc = WindowProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

if (!RegisterClassEx(&wcex))

return 0;

/\* create main window \*/

hwnd = CreateWindowEx(0,

"GLSample",

"OpenGL Sample",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT,

CW\_USEDEFAULT,

256,

256,

NULL,

NULL,

hInstance,

NULL);

ShowWindow(hwnd, nCmdShow);

/\* enable OpenGL for the window \*/

EnableOpenGL(hwnd, &hDC, &hRC);

/\* program main loop \*/

while (!bQuit)

{

/\* check for messages \*/

if (PeekMessage(&msg, NULL, 0, 0, PM\_REMOVE))

{

/\* handle or dispatch messages \*/

if (msg.message == WM\_QUIT)

{

bQuit = TRUE;

}

else

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

else

{

/\* OpenGL animation code goes here \*/

glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glPushMatrix();

glRotatef(theta, 0.0f, 0.0f, 1.0f);

glBegin(GL\_TRIANGLES);

glColor3f(1.0f, 0.0f, 0.0f); glVertex2f(0.0f, 1.0f);

glColor3f(0.0f, 1.0f, 0.0f); glVertex2f(0.87f, -0.5f);

glColor3f(0.0f, 0.0f, 1.0f); glVertex2f(-0.87f, -0.5f);

glEnd();

glPopMatrix();

SwapBuffers(hDC);

theta += 1.0f;

Sleep (1);

} }

/\* shutdown OpenGL \*/

DisableOpenGL(hwnd, hDC, hRC);a

/\* destroy the window explicitly \*/

DestroyWindow(hwnd);

return msg.wParam;

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_CLOSE:

PostQuitMessage(0);

break;

case WM\_DESTROY:

return 0;

case WM\_KEYDOWN:

{

switch (wParam)

{

case VK\_ESCAPE:

PostQuitMessage(0);

break;

}

}

break;

default:

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

return 0;

}

void EnableOpenGL(HWND hwnd, HDC\* hDC, HGLRC\* hRC)

{

PIXELFORMATDESCRIPTOR pfd;

int iFormat;

/\* get the device context (DC) \*/

\*hDC = GetDC(hwnd);

/\* set the pixel format for the DC \*/

ZeroMemory(&pfd, sizeof(pfd));

pfd.nSize = sizeof(pfd);

pfd.nVersion = 1;

pfd.dwFlags = PFD\_DRAW\_TO\_WINDOW |

PFD\_SUPPORT\_OPENGL | PFD\_DOUBLEBUFFER;

pfd.iPixelType = PFD\_TYPE\_RGBA;

pfd.cColorBits = 24;

pfd.cDepthBits = 16;

pfd.iLayerType = PFD\_MAIN\_PLANE;

iFormat = ChoosePixelFormat(\*hDC, &pfd);

SetPixelFormat(\*hDC, iFormat, &pfd);

/\* create and enable the render context (RC) \*/

\*hRC = wglCreateContext(\*hDC);

wglMakeCurrent(\*hDC, \*hRC);

}

void DisableOpenGL (HWND hwnd, HDC hDC, HGLRC hRC)

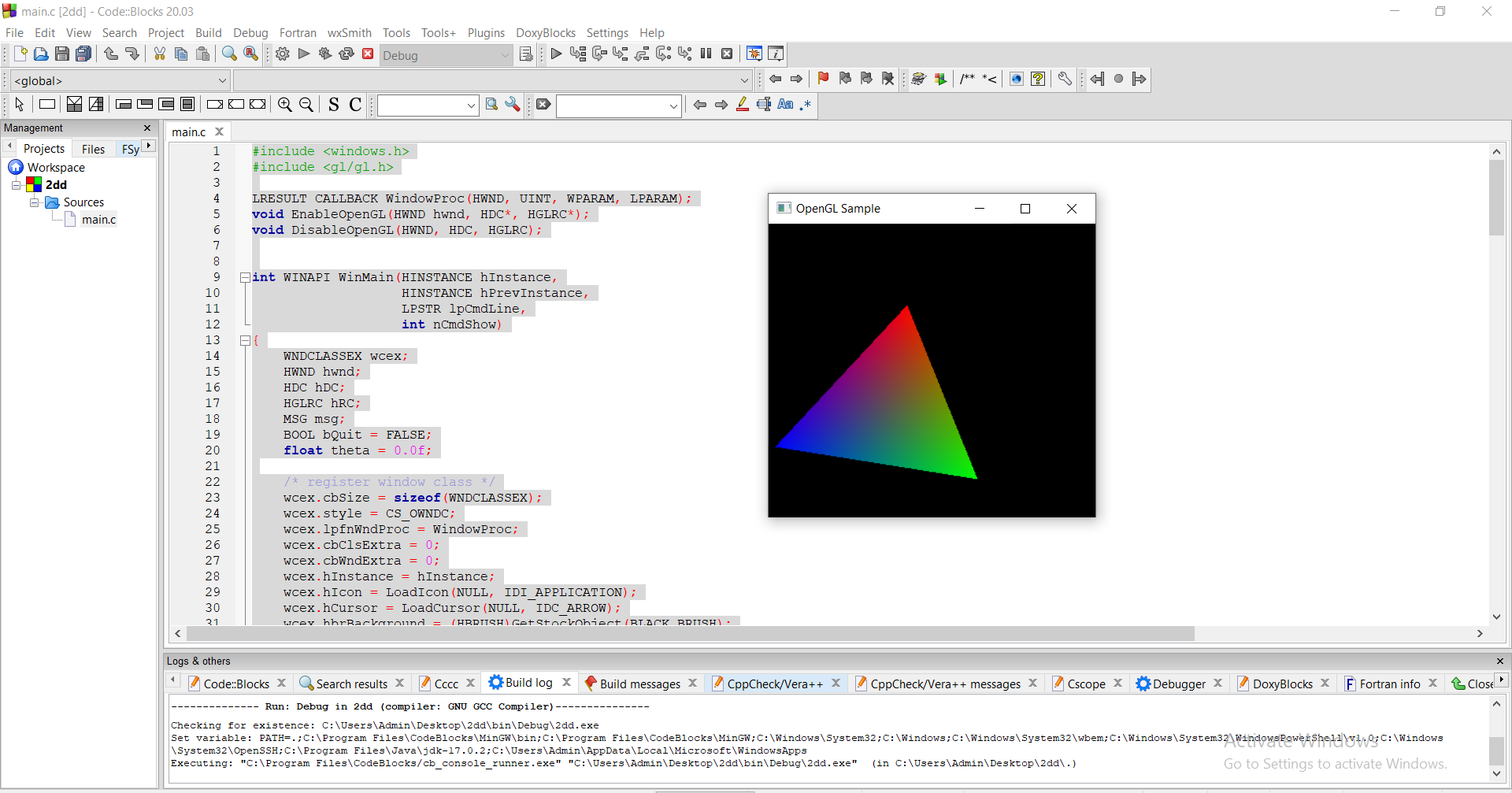
{

wglMakeCurrent(NULL, NULL);

wglDeleteContext(hRC);

ReleaseDC(hwnd, hDC);

}



LIMITATIONS:

* Limited imagination.
* Lack of simplicity .
* 2D is older less interesting, and less sought out than animation. It is the less desirable of the two technologies and, despite the quick learning time, can take more time to produce in entirety than 3D since the program 2D is built on is less advanced.

REFERENCES:

* <https://github.com/rubenandrebarreiro/2d-rotative-shapes-dance>
* <https://github.com/BhaskarVeduruparthi/2D-Rotation-of-a-Wheel/blob/main/computergraphicsproject.cpp>