# CHAPTER 1

# INTRODUCTION

## INTRODUCTION TO COMPUTER GRAPHICS

Computer Graphics is concerned with all aspects of producing pictures or images using a computer. Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television.

### Applications of Computer Graphics

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

### The Graphics Architecture

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



Memory

Frame buffer

Processor

Output Device

Input

Input

Mouse

Mouse

Input

Mouse

Device

Input

Mouse

### Figure 1.1: Components of Graphics Architecture and their working

Figure 1.1 shows the flow of information through components of a computer during execution of graphics application. The processor takes the input from all the supported input devices. It can access the memory to get additional information about the current frame and previously displayed frames. Using all the available input, the frame buffer is filled. The output device gets its instruction from frame buffer.

* 1. **INTRODUCTION TO OPENGL**

OpenGL is software used to implement computer graphics. The structure of OpenGL is similar to that of most modern APIs including Java 3D and DirectX. OpenGL is easy to learn, compared with other.

APIs are nevertheless powerful. It supports the simple 2D and 3D programs. It also supports the advanced rendering techniques. OpenGL API explains following 3 components

1. Graphics functions
2. Graphics pipeline and state machines
3. The OpenGL interfaces

There are so many polygon types in OpenGL like triangles, quadrilaterals, strips and fans. There are 2 control functions, which will explain OpenGL through,

1. Interaction with window system
2. Aspect ratio and viewports

**Frame Buffer**

**GL**

**OpenGL Application Program**

**GLUT**

**GLX**

**Xlib, Xtk**

### Figure 1.2: OpenGL Library organization

**OpenGL Application Program**

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. This ordering, as shown in Figure 1.2, is not a strict rule of how OpenGL is implemented but provides a reliable guide for predicting what OpenGL will do. The following diagram (fig 1.3) shows the assembly line approach, which OpenGL takes to process data. Geometric data (vertices, lines, and polygons) follow the path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are treated differently for part of the process. Both types of data undergo the same final steps before the final pixel data is written into the framebuffer.

Pre-vertex operations and primitive assembly

Display list

Evaluators

Per-Fragment operation

Rasterization

Pixel operation

Texture

**GLU**

Frame Buffer



Vertex data

Pixel data

### Figure 1.3: OpenGL Order of Operations

**CHAPTER 2**

**REQUIREMENTS SPECIFICATION**

**2.1 SOFTWAREREQUIREMENTS**

* + - Operating system – Windows10
    - Microsoft Visual Studio2017
    - OPENGL library files – GL, GLU, GLUT
    - Language used is C/C++

## 2.2 HARDWAREREQUIREMENTS

* + - Processor – Pentium Pro
    - Memory - 128MBRAM
    - 20GB Hard Disk Drive
    - Mouse or another pointing device
    - Keyboard
    - Display device

**CHAPTER 3**

**SYSTEM DEFINITION**

## PROJECT DESCRIPTION

A number of different sources were thoroughly searched for reference material. The major source being the **internet**. The main idea was derived from our Lab program to create a color cube and rotates it. The presentation slides by our teachers and our college lab manual containing the details of all the functions were of great help.

To create a star like, we have used two types of figure,

1. Based on a cube
2. Based on a dodecahedron

A cube is a solid figure with six square faces and eight vertices.

A dodecahedron is a solid figure with 12 equivalent pentagonal faces and 20 vertices.

* 1. **BUILT-IN FUNCTIONS**
* **void glMatrixMode(GL\_PROJECTION):** Applies subsequent matrix operations to the projection matrix stack.
* **void glutInit():** It initializes GLUT.
* **void glutlnitDisplayMode( ):** It requests a display with the properties in mode.
* **void glutCreateWindow( )**: It creates a window on the display.
* **void glutMainloop( )**: It causes the program to enter an event processing loop.
* **void glutPostRedisplay( ):** It requests that the display callback be to be executed after the current callback returns.
* **void glutDisplayFunc( ):** It registers the display function that is executed when the window needs to be redrawn.
* **void glutKeyboardFunc( ):**  It registers the keyboard callback function.
* **void glEnable( )**: It enables an openGL feature.
* **void glFlush( )**:Forces any buffered openGL commands to execute.
* **void glClearColor( )**: It sets the present RGB clearing the color buffer.
* **void glBegin( )**: It initiates new primitive and vertices.
* **void glEnd( )**: It terminates a list of vertices.

# 3.3 ALGORITHM

### STEP1: Starts the program from main ( ) function.

STEP 2: From the main function ( ) it will call the display function.

STEP 3: In the display function ( ), checks the value of i and j and accordingly draws the pyramids and then calls the redraw function ( ).

STEP 4: After the display function, the idle function, spinner( ) is called which uses some random defined functions to achieve a random rotation.

STEP 5: The idle function keep running uninterrupted till any mouse buttons are not pressed. For a mouse button interruption, the mouse ( ) function is executed which alters the values of “i” and “j” and new figure is redisplayed.

* 1. **SYSTEM ARCHITECTURE**

It shows the simple work architecture of Dancing Star.



START

INITIALIZE LIBRARY AND FUNCTIONS

MAIN



MAIN SCREEN

Read Input from Mouse

LEFT MOUSE BUTTON ADD ONE EXTRA GOMETRIC

RIGHT MOUSE BUTTON DELETES IT

END OF INPUT

GRAPHICAL OUTPUT

STOP

MOUSE

**CHAPTER 4**

**IMPLEMENTATION**

## 4.1 SOURCE CODE

# #include <stdlib.h>

# #include<math.h>

# #include <glut.h>

# float i = .2;

# int j = 0, k = 0, l = 0;

# GLfloat vertices[][3] =

# {

# {-42.53,0,57.45},{-13.14,-40.45,57.45},

# {34.41,-25,57.45},{34.41,25,57.45},

# {-13.41,40.45,57.45},{-65.50,0,13},

# {-20,-60,13},{52.89,-38.47,13},

# {52.89,38.47,13},{-20.30,62.25,13},

# {-52.89,-38.47,-13},{20,-60,-13},

# {65.50,0,-13},{20.30,62.25,-13},

# {-52.89,38.47,-13},{-34.41,-25,-57.45},

# {13.14,-40.45,-57.45},{42.53,0,-57.45},

# {13.14,40.45,-57.45},{-34.41,25,-57.45},

# {-76.08,-55.28,46.08},{29.08,-89.49,46.08},

# {94.05,0,46.08},{29.08,89.49,46.08},

# {-76.08,55.28,46.08},{-29.09,-89.53,-46.08},

# {76.08,-55.28,-46.08},{76.08,55.28,-46.08},

# {-29.08,89.49,-46.08},{-94.05,0,-46.08},

# {0,0,107.42},{0,0,-107.42},

# {-50.0,-50.0,-50.0},{50.0,-50.0,-50.0},

# {50.0,50.0,-50.0},{-50.0,50.0,-50.0},

# {-50.0,-50.0,50.0}, {50.0,-50.0,50.0},

# {50.0,50.0,50.0},{-50.0,50.0,50.0},

# {0,0,100},{0,0,-100},

# {-100,0,0},{0,100,0},

# {100,0,0},{0,-100,0}

# };

# GLfloat colors[][3] =

# {

# {0.0,0.0,1.0},{0.0,1.0,0.0},

# {0.5,1.0,0.0}, {0.0,1.0,1.0}, {1.0,0.0,0.0},

# {1.0,0.0,1.0}, {1.0,1.0,0.0}, {0.5,1.0,1.0},

# {1.0,1.0,0.50},{0.50,0.30,1.0},{0.0,1.0,.50},

# {1.0,0.0,0.0},{0.0,1.0,.50},{1.0,0.0,0.0},

# {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},

# {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0},

# {1.0,1.0,1.0},{0.0,1.0,1.0},{0.0,1.0,.50},

# {1.0,0.0,0.0},{0.0,1.0,.50},{1.0,0.0,0.0},

# {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},

# {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0},

# {1.0,1.0,1.0},{0.0,1.0,1.0},{0.0,1.0,.50},

# {1.0,0.0,0.0},{0.0,1.0,.50},{1.0,0.0,0.0},

# {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},

# {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0},

# {1.0,1.0,1.0}, {0.0,1.0,1.0},{0.0,1.0,.50},

# {1.0,0.0,0.0}

# };

# GLfloat normals[][3] = { { 0.0, 0.0, 0.0 }, { 0.0, 0.0, 0.0 }, { 0.0, 0.0, 0.0 } };

# void polygon(int a, int b, int c)

# {

# glBegin(GL\_POLYGON);

# glColor3fv(colors[a]);

# glNormal3fv(normals[a]);

# glVertex3fv(vertices[a]);

# glColor3fv(colors[b]);

# glNormal3fv(normals[b]);

# glVertex3fv(vertices[b]);

# glColor3fv(colors[c]);

# glNormal3fv(normals[c]);

# glVertex3fv(vertices[c]);

# glEnd();

# }

# void pent0()

# {

# polygon(1, 0, 31);

# polygon(2, 1, 31);

# polygon(3, 2, 31);

# polygon(4, 3, 31);

# polygon(0, 4, 31);

# }

# void pent1()

# {

# polygon(15, 16, 30);

# polygon(16, 17, 30);

# polygon(17, 18, 30);

# polygon(18, 19, 30);

# polygon(19, 15, 30);

# }

# void pent2()

# {

# polygon(1, 0, 27);

# polygon(0, 5, 27);

# polygon(5, 10, 27);

# polygon(10, 6, 27);

# polygon(6, 1, 27);

# }

# void pent3()

# {

# polygon(2, 1, 28);

# polygon(1, 6, 28);

# polygon(6, 11, 28);

# polygon(11, 7, 28);

# polygon(7, 2, 28);

# }

# void pent4()

# {

# polygon(3, 2, 29);

# polygon(2, 7, 29);

# polygon(7, 12, 29);

# polygon(12, 8, 29);

# polygon(8, 3, 29);

# }

# void pent5()

# {

# polygon(4, 3, 25);

# polygon(3, 8, 25);

# polygon(8, 13, 25);

# polygon(13, 9, 25);

# polygon(9, 4, 25);

# }

# void pent6()

# {

# polygon(0, 4, 26);

# polygon(4, 9, 26);

# polygon(9, 14, 26);

# polygon(14, 5, 26);

# polygon(5, 0, 26);

# }

# void pent7()

# {

# polygon(0, 1, 27);

# polygon(0, 5, 27);

# polygon(5, 10, 27);

# polygon(10, 6, 27);

# polygon(6, 1, 27);

# }

# void pent8()

# {

# polygon(15, 16, 23);

# polygon(16, 11, 23);

# polygon(11, 6, 23);

# polygon(6, 10, 23);

# polygon(10, 15, 23);

# }

# void pent9()

# {

# polygon(16, 17, 24);

# polygon(17, 12, 24);

# polygon(12, 7, 24);

# polygon(7, 11, 24);

# polygon(11, 16, 24);

# }

# void pent10()

# {

# polygon(17, 18, 20);

# polygon(18, 13, 20);

# polygon(13, 8, 20);

# polygon(8, 12, 20);

# polygon(12, 17, 20);

# }

# void pent11()

# {

# polygon(18, 19, 21);

# polygon(19, 14, 21);

# polygon(14, 9, 21);

# polygon(9, 13, 21);

# polygon(13, 18, 21);

# }

# void pent12()

# {

# polygon(19, 15, 22);

# polygon(15, 10, 22);

# polygon(10, 5, 22);

# polygon(5, 14, 22);

# polygon(14, 19, 22);

# }

# void cube1()

# {

# polygon(32, 33, 40);

# polygon(33, 34, 40);

# polygon(34, 35, 40);

# polygon(35, 32, 40);

# }

# void cube2()

# {

# polygon(37, 33, 42);

# polygon(33, 34, 42);

# polygon(34, 38, 42);

# polygon(37, 38, 42);

# }

# void cube3()

# {

# polygon(36, 39, 44);

# polygon(39, 35, 44);

# polygon(35, 32, 44);

# polygon(32, 36, 44);

# }

# void cube4()

# {

# polygon(33, 32, 43);

# polygon(33, 37, 43);

# polygon(36, 37, 43);

# polygon(36, 32, 43);

# }

# void cube5()

# {

# polygon(36, 37, 41);

# polygon(37, 38, 41);

# polygon(38, 39, 41);

# polygon(39, 36, 41);

# }

# void cube6()

# {

# polygon(34, 35, 45);

# polygon(35, 39, 45);

# polygon(39, 38, 45);

# polygon(38, 34, 45);

# }

# static GLfloat theta[] = { 0.10,0.10,0.10 };

# static GLint axis = 2;

# void display(void)

# {

# glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

# glLoadIdentity();

# glRotatef(theta[0], 1.0, 0.0, 0.0);

# glRotatef(theta[1], 0.0, 0.0, 1.0);

# glRotatef(theta[2], 0.0, 1.0, 0.0);

# if (j <= 0)

# {

# if (l > 0)

# cube1();

# if (l > 1)

# cube5();

# if (l > 2)

# cube2();

# if (l > 3)

# cube3();

# if (l > 4)

# cube4();

# if (l > 5)

# cube6();

# }

# else

# {

# if (j > 0)

# pent0();

# if (j > 1)

# pent1();

# if (j > 2)

# pent2();

# if (j > 3)

# pent3();

# if (j > 4)

# pent4();

# if (j > 5)

# pent5();

# if (j > 6)

# pent6();

# if (j > 7)

# pent7();

# if (j > 8)

# pent8();

# if (j > 9)

# pent9();

# if (j > 10)

# pent10();

# if (j > 11)

# pent11();

# if (j >= 11)

# pent12();

# }

# glFlush();

# glutSwapBuffers();

# }

# void spinner()

# {

# theta[axis] = theta[axis] + 0.8\*sin(i);

# if (k == 250 || sin(i) == 0)

# {

# axis = (axis + 1) % 3;

# if (k == 250 || (sin(i) >= .5998 && sin(i) <= 1))

# k = 0;

# }

# if (theta[axis] >= 360.0)

# {

# theta[axis] -= 360.0;

# }

# i += .005;

# if (i > 6.2830)

# i = 0;

# k++;

# glutPostRedisplay();

# }

# void mouse(int btn, int state, int x, int y)

# {

# if (j != 12 && l != 7)

# if (btn == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)

# {

# if (l < 6)

# l++;

# else

# j++;

# }

# if (btn == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN) axis = 1;

# if (btn == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)

# {

# if (j > 0)

# j--;

# else if (l > 0)

# l--;

# }

# }

# void myReshape(int w, int h)

# {

# glViewport(0, 0, w, h);

# glMatrixMode(GL\_PROJECTION);

# glLoadIdentity();

# if (w <= h)

# glOrtho(-150.0, 150.0, -150.0 \* (GLfloat)h / (GLfloat)w,

# 150.0 \* (GLfloat)h / (GLfloat)w, -750.0, 750.0);

# else

# glOrtho(-150.0 \* (GLfloat)w / (GLfloat)h,

# 150.0 \* (GLfloat)w / (GLfloat)h, -150.0, 150.0, -750.0, 750.0);

# glMatrixMode(GL\_MODELVIEW);

# }

# void main(int argc, char \*\*argv)

# {

# glutInit(&argc, argv);

# glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

# glutInitWindowSize(1250, 750);

# glutCreateWindow("Dancing Star");

# glutReshapeFunc(myReshape);

# glutDisplayFunc(display);

# glutIdleFunc(spinner);

# glutMouseFunc(mouse);

# glEnable(GL\_DEPTH\_TEST);

# glutMainLoop();

# }

# CHAPTER 5

**TESTING AND RESULTS**

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

* 1. **TESTING PROCESS**

Testing is an integral part of software development. Testing process certifies whether the product that is developed compiles with the standards that it was designed to. Testing process involves building of test cases against which the product has to be tested.

* 1. **TESTING OBJECTIVES**

The main objectives of testing process are as follows.

* Testing is a process of executing a program with the intent of finding an error.
* A good test case is one that has high probability of finding undiscovered error.
* A successful test is one that uncovers the undiscovered error.
  1. **DIFFERENT TYPES OF TESTING**

### Unit Testing

Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components.

### Module Testing

A module is a collection of dependent components such as an object class, an abstract Data type or some looser collection of procedures and functions. A module related Components, so can be tested without other system modules.

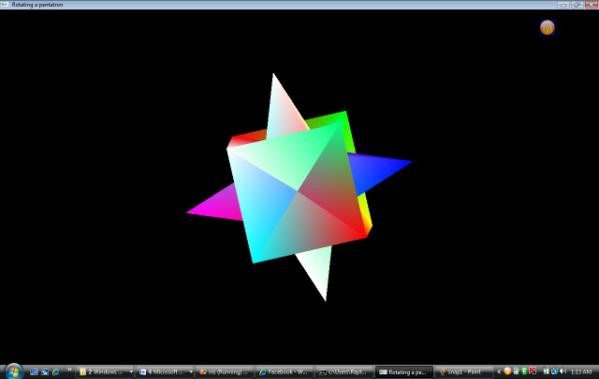
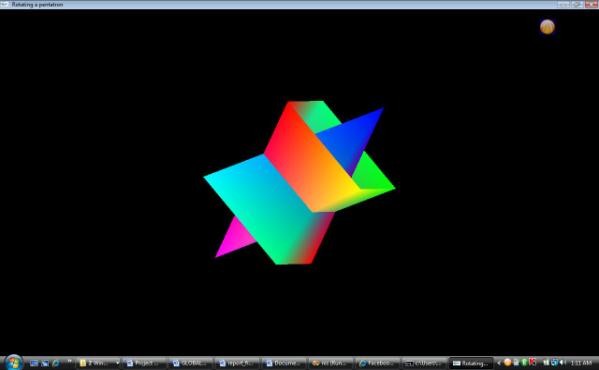
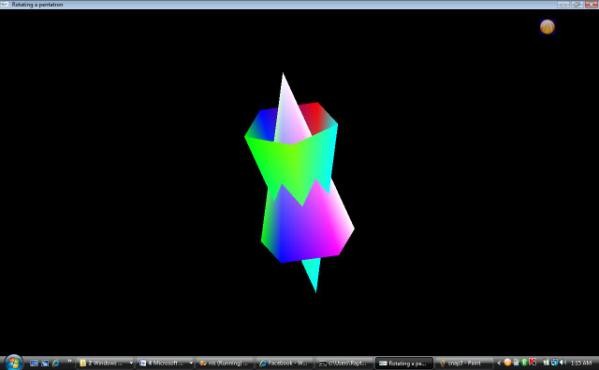
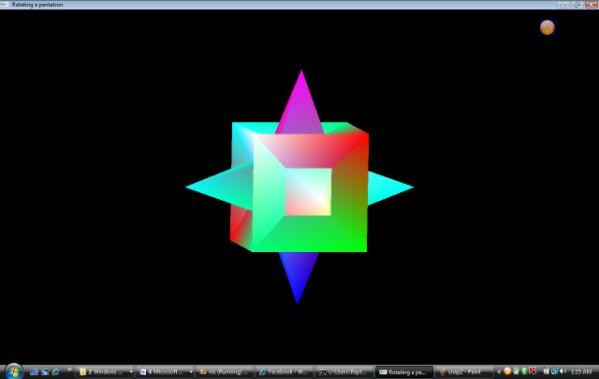
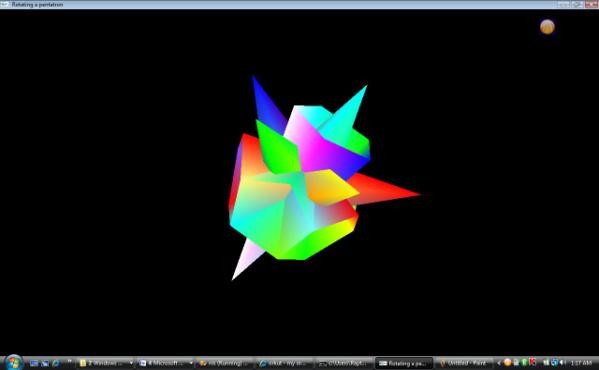
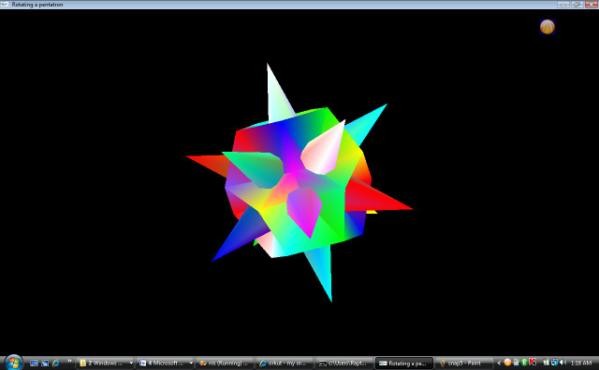
### System Testing

This is concerned with finding errors that result from unanticipated interaction between Sub-system interface problems.

### Acceptance Testing

The system is tested with data supplied by the system customer rather than simulated test data.

**CHAPTER 6**

**** **SNAPSHOTS OF THE OUTPUT SCREEN**

**CONCLUSION**

We had a great experience in the course of designing this package, which made us discover and learn many things pertaining to the topic of OpenGL programming. I have tried to my best potential to incorporate all the basic level requirements of a normal graphics package for Windows operating system.



This is very reliable graphics package supporting various primitive objects like polygon, line loops, etc. Also colour selection, menu, keyboard and mouse-based interface are included. Transformations like translation, rotation, scaling is also provided.



Owing to certain constraints I could not incorporate all the tasks that a graphics package should perform. However, it meets the basic requirements of user successfully. Since its user friendly it enables the user to interact effectively and easily.

Special attention has been provided to the interfaces and menus that make its use comfortable. I hope this package proves to be flexible in all respects to one and all.

## BIBLIOGRAPHY

## Books

* Edward Angel, ―Interactive Computer Graphics, 5th edition, Pearson Education, 2005
* Jackie L. Neider, Mark Warhol, Tom R. Davies, OpenGL Red Book, 2nd Revised Edition, 2005
* OpenGL Super Bible (5th Edition) by Richard S. Wright
* OpenGL Programming Guide (7th edition) by Dave Shreiner
* OpenGL 4 shading language cookbook (2nd edition) by David Wolff

**Websites**

* [http://www.opengl.org](http://www.opengl.org/)/code/C17
* [http://](http://en.wikipedia.org/wiki/OpenGL)[www.opengl-tutorial.org/](http://www.opengl-tutorial.org/)
* [http://basic4gl.wikispaces.com/3D+Drawing+in+openGL](http://basic4gl.wikispaces.com/3D%2BDrawing%2Bin%2BopenGL)
* [http://](http://en.wikipedia.org/wiki/OpenGL)[www.glprogramming.com/red/](http://www.glprogramming.com/red/)
* [http://](http://en.wikipedia.org/wiki/OpenGL)learnopengl.com/