Contents

Chapter 1

INTRODUCTION

1.1 Computer Graphics

- Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D Or 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly.
- Computers have become a powerful medium for the rapid and economical production of pictures.
- Graphics provide a so natural means of communicating with the computer that they have become widespread.
- Interactive graphics is the most important means of producing pictures since the invention of photography and television .
- We can make pictures of not only the real world objects but also of abstract objects such as mathematical surfaces on 4D and of data that have no inherent geometry.
- A computer graphics system is a computer system with all the components of the general purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.

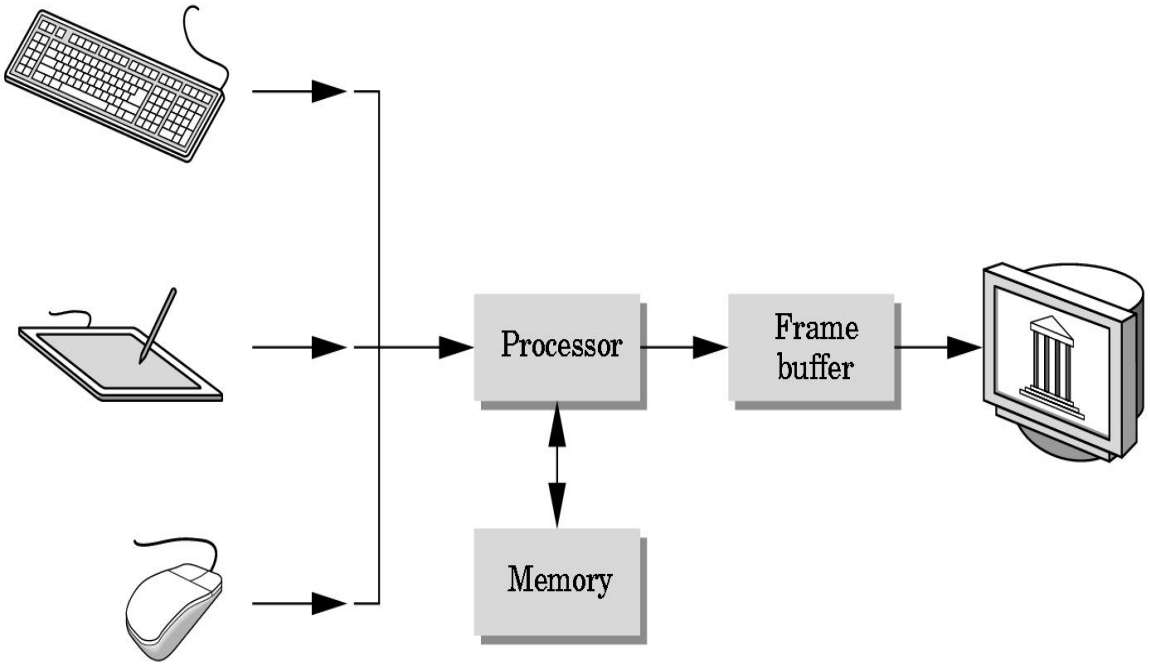

Fig:-(a) Computer Graphics System

1.2 OpenGL Technology

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms.

OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

OpenGL Available Everywhere: Supported on all UNIX® workstations, and shipped standard with every Windows 95/98/2000/NT and MacOS PC, no other graphics API operates on a wider range of hardware platforms and software environments.

OpenGL runs on every major operating system including Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT, Linux, Open Step, and BeOS; it also works with every major windowing system, including Win32, MacOS, Presentation Manager, and X-Window System. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl and Java and offers complete independence from network protocols and topologies.

The OpenGL interface

Our application will be designed to access OpenGL directly through functions in three libraries namely: gl,glu,glut.

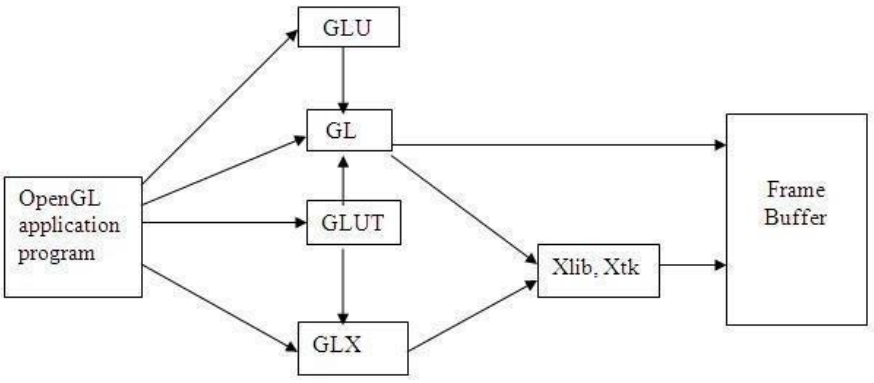

Fig:(b) OpenGL Interface

Chapter 2

LITERATURE OVERVIEW

The basic functions like glcolor3f(…..); glrotatef(…..); gltranslate(…..) etc. that are most commoly Used in the code are taken form the prescribed VTU text book “INTERACTIVE COMPUTER GRAPHICS” 5th edition by Edward Angel.[1].

The lab programs in the syllabus also serve as a basic template for creating a project. The usage of colors and specifications are taken from the various programs that were taught in the lab. [1]. The VTU prescribed text book serves as a huge database of functions and they are used in the project.

The C concepts which are used, are being taken from the book named Yeshwant Kanitkar. Some concepts like constructing bowl and fountain are taken from the search results in the codecolony.com.

Chapter 3

REQUIREMENTS AND SPECIFICATIONS

3.1 Purpose of the requirements document

The software requirement specification is the official statement of what is required for development of particular project. It includes both user requirements and system requirements. This requirement document is utilized by variety of users starting from project manager who gives project to the engineer responsible for development of project.

It should give details of how to maintain, test, verify and what all the actions to be carried out through life cycle of project.

Scope of the project

The scope is to use the basic primitives defined in openGL library creating complex objects. We make use of different concepts such as pushmatrix(),translate() ,popmatrix(),timer function.

Definition

The project ***CATCH ME IF YOU CAN!!!*** is created to demonstrate OpenGL's concepts. It encompasses some of the skills learnt in our OpenGL classes such as pushmatrix(),translate() ,popmatrix(),timer function

.

Acronyms & Abbreviations

OpenGL provides a powerful but primitive set of rendering command, and all higher level design must be done in terms of these commands.

OpenGL Utility Toolkit(GLUT):- windows-system-independent toolkit.

References

OpenGL tutorials

Interactive Computer Graphics(Edward Angel)

3.2 Specific requirements

User Requirement:

- Easy to understand and should be simple.
- The built-in functions should be utilized to maximum extent.
- OpenGL library facilities should be used.

Software Requirements:

- Ubuntu Os.
- Eclipse compiler.

Hardware Requirements:

- Processor- Intel or AMD(Advanced Micro Devices)
- RAM- 512MB(minimum)
- Hard Disk-1MB(minimum)
- Mouse
- Keyboard
- onitor

Chapter 4

DESIGN

4.1 User Defined Functions

- *myinit():*

his function initializes light source for ambient, diffuse and specular types.

- *display():*

This function creates and translates all the objects in a specified location in a particular order and also rotates the objects in different axes.

```
glClear(GL_COLOR_BUFFER_BI
        T);glFlush();
```

- **merfunc():**
  This function starts a timer in the event loop that delays the event loop for delay miiliseconds.

- *MainLoop():*

This function whose execution will cause the program to begin an event processing loop.

- *PushMatrix():*

ave the present values of attributes and matrices placing ,or pushing on the top of the stack.

- *PopMatrix():*

We can recover them by removing them from stack;

- *Translated();*

In translate func the variables are components of the displacement vector.

- *main():*

The execution of the program starts from this function. It initializes the graphics system and includes many callback functions.

- *PostRedisplay():*

It ensures that the display will be drawn only once each time the program goes through the event loop.

Chapter 5

IMPLEMENTATION

5.1 FUNCTIONS

GL_LINES -

Treats each pair of vertices as an independent line segment.
Vertices 2n - 1 and 2n define line n. N/2 lines are drawn.

GL_LINE_LOOP -

Draws a connected group of line segments from the first vertex to the last, then back to the first. Vertices n and n + 1 define line n. The last line, however, is defined by vertices N and N lines are drawn.

Basic Functions

glPushMatrix, glPopMatrix Function

The glPushMatrix and glPopMatrix functions push and pop the current matrix stack.

SYNTAX: void glPushMatrix();
void glPopMatrix(void);

glBegin, glEnd Function

The glBegin and glEnd functions delimit the vertices of a primitive or a group of like primitives.

SYNTAX:
void glBegin, glEnd(GLenum mode);

PARAMETERS:

- mode -

The primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd. The following are accepted symbolic constants and their meanings:

Transformation Functions

glTranslate Function

The glTranslated and glTranslatef functions multiply the current matrix by a translation matrix.

SYNTAX:
void glTranslate(x, y, z);

PARAMETERS:

- x, y, z - The x, y, and z coordinates of a translation vector.

Funtions used to display

glMatrixMode

Function

The glMatrixMode function specifies which matrix is the current matrix.

SYNTAX:
void glMatrixMode(GLenum mode);

PARAMETERS:

❖ mode - The matrix stack that is the target for subsequent matrix operations. The mode parameter can assume one of three values:

Value	Meaning
GL_MODELVIEW	Applies subsequent matrix operations to the modelview matrix stack.

glLoadIdentity Function

The glLoadIdentity function replaces the current matrix with the identity matrix.

SYNTAX:

void glLoadIdentity(void);

5.2 FUNCTIONS USED TO SET THE VIEWING VOLUME

glOrtho

This function defines orthographic viewing volume with all parameters measured from the centre of projection.

multiply the current matrix by a perspective matrix.

SYNTAX:

void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)

PARAMETERES:

- left, right -

Specify the coordinates for the left and right vertical clipping planes.

- bottom, top -

Specify the coordinates for the bottom and top horizontal clipping planes.

- nearVal, farVal -

Specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.

5.3 LL BACK FUNCTIONS

glutDisplayFunc Function

glutDisplayFunc sets the display callback for the current window.

SYNTAX:
void glutDisplayFunc(void (*func)(void));

glutReshapeFunc Function

glutReshapeFunc sets the reshape callback for the current window.

SYNTAX:
void glutReshapeFunc(void (*func)(int width, int height));

5.4 MAIN FUNCTION

glutInit Function

glutInit is used to initialize the GLUT library.

SYNTAX:

glutInit(int *argcp, char **argv);

PARAMETERS:

- argcp - A pointer to the program's unmodified argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.
- Argv -
  The program's unmodified argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.
  - glutInit(&argc,argv);

glutInitDisplayMode Function

glutInitDisplayMode sets the initial display mode.

SYNTAX:
void glutInitDisplayMode(unsigned int mode);

PARAMETERS:

- mode -
  Display mode, normally the bitwise OR-ing of GLUT display mode bit masks. See values below:

GLUT_RGB: An alias for GLUT_RGBA.
GLUT_DOUBLE:Bit mask to select a double buffered window. This overrides GLUT_SINGLE.
GLUT_DEPTH: Bit mask to select a window with a depth buffer.

glutMainLoop Function

glutMainLoop enters the GLUT event processing loop.

SYNTAX:

void glutMainLoop(void);

Chapter-6

APPENDIX -A

SOURCE CODE

```
#include <GL/glut.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#define GL_SILENCE_DEPRECATION

GLdouble PI = 3.14159265;
GLfloat yMove = 50, xMove = 10;
GLfloat ySteps = 2, xSteps = 1;
int yFlag = 1, xFlag = 1, animateFlag = 1;

//TRIGGER ANIMATION
void Animate() {
   if(animateFlag == 1) {
      //Animate  about  y-axis
      if(yMove > -50 && yFlag == 1) {
         yMove -= ySteps;
         if(yMove <= -50) {
            yFlag = 0;
         }
      }
      if(yMove < 50 && yFlag == 0) {
         yMove += ySteps;
         if(yMove >= 50) {
            yFlag = 1;
         }
      }

      //Animate  about  x-axis
      if(xMove > -10 && xFlag == 1) {
         xMove -= xSteps;
         if(xMove <= -10) {
            xFlag = 0;
         }
      }
      if(xMove < 10 && xFlag == 0) {
         xMove += xSteps;
         if(xMove >= 10) {
            xFlag = 1;
         }
      }

      glutPostRedisplay(); //Redraw the screen
   }
}//SKY
 void DrawSky() {
```

```
    glColor3f(0.53f, 0.81f, 0.92f);
    glBegin(GL_QUADS);
        glVertex2i(0, 500);
        glVertex2i(500, 500);
        glVertex2i(500, 250);
        glVertex2i(0, 250);
    glEnd();
}

//SUN
void DrawSun() {
    glColor3f(1.0, 0.5, 0);
    glBegin(GL_POLYGON);
        GLdouble angle, x, y;
        GLdouble radianConversion = PI / 180;
        for (int i = 0; i < 360; i++) {
            angle = i * radianConversion;
            x = 450 + 50 * cos(angle);
            y = 450 + 50 * sin(angle);
            glVertex2d(x, y);
        }
    glEnd();
}

//CLOUDS
void DrawEllipse(float x, float y, float xRadius, float yRadius) {
    glBegin(GL_POLYGON);
    GLdouble radianConversion = PI / 180;
    for (int i = 0; i < 360; i++) {
        float angle = i * radianConversion;
        float xPos = x + xRadius * cos(angle);
        float yPos = y + yRadius * sin(angle);
        glVertex2f(xPos, yPos);
    }
    glEnd();
}
void DrawClouds(int numberOfClouds) {
    glColor3f(1, 1, 1);
    srand(time(NULL));
    for (int i = 0; i < numberOfClouds; i++) {
        float xPos = rand() % 500;
        float yPos = 300 + (rand() % 150);
        float xRadius = 20 + (rand() % 30);
        float yRadius = 10 + (rand() % 20);

        for (int j = 0; j < 3; j++) {
            DrawEllipse(xPos + j * xRadius / 2, yPos, xRadius, yRadius);
        }
    }
}

//GRASSLAND
void DrawGrassland() {
```

```
    glColor3f(0.30f, 0.78f, 0.30f);
    glBegin(GL_QUADS);
        glVertex2i(0, 250);
        glVertex2i(500, 250);
        glVertex2i(500, 0);
        glVertex2i(0, 0);
        glEnd();
} //TREE
void DrawTree(GLint x, GLint y) {
    // Draw the stem
    glColor3f(0.35f, 0.16f, 0.14f);
    glBegin(GL_QUADS);
        glVertex2i(395 + x, 250 + y);
        glVertex2i(405 + x, 250 + y);
        glVertex2i(405 + x, 150 + y);
        glVertex2i(395 + x, 150 + y);
    glEnd();

    // Draw the green leaves
    glBegin(GL_TRIANGLES);
        glColor3f(0.0f, 0.4f, 0.0f);
        glVertex2i(380 + x, 200 + y);
        glVertex2i(420 + x, 200 + y);
        glVertex2i(400 + x, 300 + y);

        glColor3f(0.0f, 0.5f, 0.0f);
        glVertex2i(370 + x, 250 + y);
        glVertex2i(430 + x, 250 + y);
        glVertex2i(400 + x, 350 + y);

        glColor3f(0.0f, 0.6f, 0.0f);
        glVertex2i(360 + x, 300 + y);
        glVertex2i(440 + x, 300 + y);
        glVertex2i(400 + x, 400 + y);
    glEnd();
}

//MOUNTAINS
void DrawMountains() {
    glBegin(GL_TRIANGLE_STRIP);
        glColor3f(0.5f, 0.5f, 0.5f);
        glVertex2i(0, 250);
        glVertex2i(100, 400);
        glVertex2i(200, 250);

        glColor3f(0.55f, 0.55f, 0.55f);
        glVertex2i(200, 250);
        glVertex2i(300, 400);
        glVertex2i(400, 250);

        glColor3f(0.6f, 0.6f, 0.6f);
        glVertex2i(400, 250);
        glVertex2i(500, 400);
```

```
glVertex2i(600, 250);
glEnd();
 }

 //RIVER
 void DrawRiver() {
    glColor3f(0.25f, 0.47f, 0.87f);
    glBegin(GL_QUADS);
       glVertex2i(350, 0);
       glVertex2i(150, 0);
       glVertex2i(210, 250);
       glVertex2i(280, 250);
    glEnd();
 }

 //PODIUM
 void DrawPodium() {
    glColor3f(0.5f, 0.25f, 0.1f);
    glBegin(GL_QUADS);
       glVertex2i(80, 50);
       glVertex2i(110, 50);
       glVertex2i(110, 40);
       glVertex2i(80, 40);
    glEnd();
 }
 //MAIN DRAW FUNCTION
 void Draw() {
    GLfloat x[4], y1[4], y2[4], y3[4], y4[4];
    glClear(GL_COLOR_BUFFER_BIT);

    /* Initialize the control points */
    x[0] = 100; x[1] = 200; x[2] = 200;  x[3] = 300 - xMove;
    y1[0] = 450; y1[1] = 450 + yMove; y1[2] = 450 - yMove; y1[3] = 450; //Upper boundary of orange
    y2[0] = 400; y2[1] = 400 + yMove; y2[2] = 400 - yMove; y2[3] = 400; //Upper boundary of white
    y3[0] = 350; y3[1] = 350 + yMove; y3[2] = 350 - yMove; y3[3] = 350; //Upper boundary of green
    y4[0] = 300; y4[1] = 300 + yMove; y4[2] = 300 - yMove; y4[3] = 300; //Lower boundary of green

    /* Bézier curve algorithm to determine the intermediate points */
    GLdouble xt[200], y1t[200], y2t[200], y3t[200], y4t[200], t;
    int i,c; //c is the number of intermediate points
    for (i = 0, t = 0, c = 0; t <= 1; i++, t += 0.01) {
       xt[i] = pow(1 - t, 3) * x[0] + 3 * t * pow(1 - t, 2) * x[1] + 3 * pow(t, 2) * (1 - t) * x[2] + pow(t, 3) *
x[3];
       y1t[i] = pow(1 - t, 3) * y1[0] + 3 * t * pow(1 - t, 2) * y1[1] + 3 * pow(t, 2) * (1 - t) * y1[2] + pow(t, 3)
* y1[3];
       y2t[i] = pow(1 - t, 3) * y2[0] + 3 * t * pow(1 - t, 2) * y2[1] + 3 * pow(t, 2) * (1 - t) * y2[2] + pow(t, 3)
* y2[3];
       y3t[i] = pow(1 - t, 3) * y3[0] + 3 * t * pow(1 - t, 2) * y3[1] + 3 * pow(t, 2) * (1 - t) * y3[2] + pow(t, 3)
* y3[3];
       y4t[i] = pow(1 - t, 3) * y4[0] + 3 * t * pow(1 - t, 2) * y4[1] + 3 * pow(t, 2) * (1 - t) * y4[2] + pow(t, 3)
* y4[3];
       c++;
    }

    /* Draw the sky */
    DrawSky();
```

```
    /* Draw the sun */
    DrawSun();

    /* Draw the mountain */
    DrawMountains();

    /* Draw clouds */
    DrawClouds(3);

    /* Draw the grassland */
    DrawGrassland();

    /* Draw the river */
    DrawRiver();

    /* Draw tree */
    DrawTree(0, 0);
    DrawTree(50, -60);

    /* Draw the podium */
    DrawPodium();

    /* Draw the flag */

    //Draw the yellow part
    glColor3f(1.0, 1.0f, 0);
    glBegin(GL_QUAD_STRIP);
       for (i = 0; i < c; i++) {
          glVertex2d(xt[i], y1t[i]);
          glVertex2d(xt[i], y2t[i]);
       }
    glEnd();

    //Draw the red part
    glColor3f(1.0f, 0, 0);
    glBegin(GL_QUAD_STRIP);
       for (i = 0; i < c; i++) {
          glVertex2d(xt[i], y2t[i]);
          glVertex2d(xt[i], y3t[i]);
       }
    glEnd();
    /* Draw the pole */
    glColor3f( 0.6f, 0.6f, 0.3f);
    glRecti(90, 460, 100, 50);

    glFlush();
}

void Menu(int n) {
    switch (n) {
       case 1:
          animateFlag = 1;
          break;
       case 2:
          animateFlag = 0;
```

```
            break;
        default:
            exit(0);
    }

    glutPostRedisplay(); //Redraw the screen
}

void MyInit() {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 500, 0, 500);
    glMatrixMode(GL_MODELVIEW);

    //Create the menu
    glutCreateMenu(Menu);
    glutAddMenuEntry("Start",1);
    glutAddMenuEntry("Stop",2);
    glutAddMenuEntry("Exit",3);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

int main(int argC, char *argV[]) {
    glutInit(&argC, argV);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutCreateWindow("Animate Flag");
    MyInit();
    glutDisplayFunc(Draw);
    glutIdleFunc(Animate);
    glutMainLoop();
    return 0;
}
```

Chapter-7

SNAP SHOTS

Conclusion

Chapter-8

The project was started with the designing phase in which I figured the requirements needed, the layout design, then comes the detail designing of each function after which, was the testing and debugging stage.

have tried to implement the project making it as user-friendly and error free as possible. We regret any errors that may have inadvertently crept in.

BIBLIOGRAPHY

[1] OpenGL Programming Guide (Addison-Wesley Publishing Company)

[2] The OpenGL Utility Toolkit (GLUT)Programming Interface
API Version 3 BY MARK J. KILGARD

[3] Interactive computer graphics
-A top don approach by using Open GL by EDWARD ANGEL