**CHAPTER 1**

**INTRODUCTION**

**1.1 Introduction to Computer Graphics**

Computer Graphics is concerned with all aspects of producing pictures or images using a computer. We can create images that are indistinguishable from photographs of real objects. In other terms, Computer Graphics re the graphics created by the computers, and more generally, the representation and manipulation of image data by a computer.

The development of computer graphics has been driven both by the needs of the user community and by advances in hardware and software.

Typically, the term Computer Graphics refers to several different things.

* The representation and manipulation of image data by a computer.
* The various technologies used to create and manipulate images.
* The images so produced, and manipulating visual content.

**1.2 History of Computer Graphics**

The phrase Computer Graphics was coined in 1960 by William Fetter, a graphic designer for Boeing. The field of Computer Graphics developed with the emergence of computer graphics hardware. Early projects like the Whirlwind and SAGE projects introduced the CRT as a viable display and interaction interface and introduced the light pen as an input device.

Further advances in computing led to greater advancements in interactive computer graphics. In 1959, the TX-2 computer was developed at MIT's Lincoln Laboratory. A light pen could be used to draw sketches on the computer using Ivan Sutherland’s revolutionary Sketchpad software.

Also in 1961 another student at MIT, Steve Russell, created the first video game, Spacewar. E. E. Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-giro gravity attitude control system" in 1963. In this computer generated film, Zajac showed how the attitude of a satellite could be altered as it orbits the Earth. Many of the most important early breakthroughs in computer graphics research occurred at the University of Utah in the 1970s.

The first major advance in 3D computer graphics was created at UU by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are "behind" the object from the viewer's perspective, and thus should be "hidden" when the computer creates (or renders) the image.

Graphics and application processing were increasingly migrated to the intelligence in the workstation, rather than continuing to rely on central mainframe and mini-computers. 3D graphics became more popular in the 1990s in gaming, multimedia and animation. Computer graphics used in films and video games gradually began to be realistic to the point of entering the uncanny valley. Examples include the later *Final Fantasy* games and animated films like *The Polar Express*.

**1.3 Applications of Computer Graphics**

The development of computer graphics has been driven both by the needsof the user community and by advances in hardware and software. The applications of computer graphics are many and varied. We can however divide them into four major areas.

* Display of information: More than 4000 years ago, the Babylonians developed floor plans of buildings on stones. Today, the same type of information is generated by architects using computers. Over the past 150 years, workers in the field of statistics have explored techniques for generating plots. Now, we have computer plotting packages. Supercomputers now allow researchers in many areas to solve previously intractable problems. Thus, Computer Graphics has innumerable applications.
* Design: Professions such as engineering and architecture are concerned with design. Today, the use of interactive graphical tools in CAD, in VLSI circuits, characters for animation have developed in a great way.
* Simulation and animation: One of the most important uses has been in pilots’ training. Graphical flight simulators have proved to increase safety and reduce expenses. Simulators can be used for designing robots, plan it’s path, etc. Video games and animated movies can now be made with low expenses.
* User interfaces: Our interaction with computers has become dominated by a visual paradigm. The users’ access to internet is through graphical network browsers. Thus Computer Graphics plays a major role in all fields.

**1.4 Introduction to OpenGL**

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that are used to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined hardware-independent interface to be implemented on many different hardware platforms.

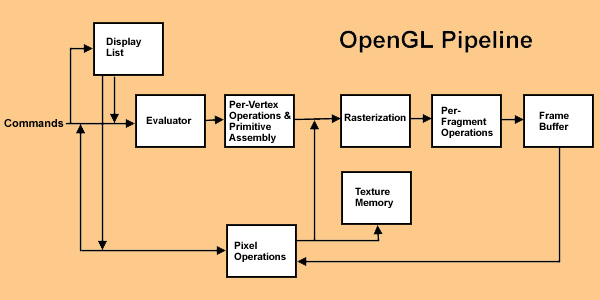
These are certain characteristics of OpenGL:

* OpenGL is a better documented API.
* OpenGL is much easier to learn and program.
* OpenGL has the best demonstrated 3D performance for any API.

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it's possible for the API to be implemented entirely in software, it's designed to be implemented mostly or entirely in hardware.

In addition to being language-independent, OpenGL is also platform-independent. The specification says nothing on the subject of obtaining, and managing, an OpenGL context, leaving this as a detail of the underlying windowing system. For the same reason, OpenGL is purely concerned with rendering, providing no APIs related to input, audio, or windowing.

OpenGL is an evolving API. New versions of the OpenGL specification are regularly released by the Khronos Group, each of which extends the API to support various new features. In addition to the features required by the core API, GPU vendors may provide additional functionality in the form of *extensions*. Extensions may introduce new functions and new constants, and may relax or remove restrictions on existing OpenGL functions. Vendors can use extensions to expose custom APIs without needing support from other vendors or the Khronos Group as a whole, which greatly increases the flexibility of OpenGL. All extensions are collected in, and defined by, the OpenGL Registry.



**1.4 OpenGL Pipeline**

**1.5 Introduction to GLUT**

GLUT is the OpenGL utility toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing API for OpenGL. GLUT makes it easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms. GLUT is designed for constructing small to medium sized OpenGL programs.

While GLUT is well-suited to learning OpenGL and developing simple OpenGLapplications, GLUT is not a full-featured toolkit so large applications requiringsophisticated user interfaces are better off using native window system toolkits.The GLUT library has both C, C++ (same as C), FORTRAN, and ADAprogramming bindings. The GLUT source code distribution is portable to nearly all OpenGL implementations and platforms.

GL

GLUT

GLX

Xlib, Xtk

Frame

buffer

OpenGL

Application

program

GLU

**1.5 library organization of OpenGL**

**1.6 Applications of OpenGL**

* OpenGL (Open Graphics Library)is a cross-language, multi-platform API for rendering 2D and 3D computer graphics.
* The API is typically used to interact with a GPU, to achieve hardware-accelerated rendering.
* It is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation, and video games.

**1.7 OpenGL primitives**

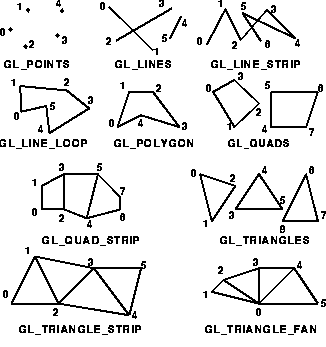
OpenGL supports two classes of primitives:

* Geometric Primitives
* Image(Raster) Primitives

Geometric primitives are specified in the problem domain and include points, line segments, polygons, curves and surfaces.

Raster primitives, such as arrays of pixels pass through a separate parallel pipeline on their way to the frame buffer.

There are ten basic OpenGL primitives:

****

**1.7 OpenGL Primitives**

**CHAPTER2**

**SYSTEM REQUIREMENTS**

**2.1 Hardware Requirements:**

* Pentium 90MHz Processor or Higher
* VGA 640x480 or higher-resolution screen supported by Microsoft Windows.
* Recommended 128 MB RAM or Higher (24 MB RAM for Windows 95/98, 32 MB for Windows NT)
* 100GB SATA (Serial Advanced Technology Attachment) Hard Drive
* 5400 RPM hard disk drive
* DirectX 9 capable video card running at 1024 x 768
* DVD-ROM Drive
* Input devices: Keyboard, Mouse.
* Output devices: Monitor.

**2.2 Software Requirements:**

* Operating System: Microsoft Windows NT 4.0 or later, or Microsoft Windows 95 or later.
* Microsoft Visual Studio 6.0
* glut.h header file
* glut.dll library files

# **Chapter3**

# **ABOUT THE PROJECT**

**3.1 Introduction to the project**

This mini projection HELICOPTER GAME displays a helicopter with all the

moving functions. A copter moves up if you hold the mouse button. If not, it keeps going down. An obstacle appears from the right and the task is to jump over it.

The mouse right button is used for moment of helicopter and also K key is used for upward movement of helicopter and M key is used for downward movement of helicopter.

Initially the player has to enter the name and the game begins.The display screen contains the moving helicopter and the obstacles and simultaneously bottom of the display contains the level and distance covered by the helicopter.

**3.2 User Defined Functions**

There are nine user defined functions in the source code of HELICOPTER GAME.

1. void display ()

It is used to display the welcome screen and exit screen. It displays and controls the movement of helicopter and objects

1. void drawcopter()

Used to create the helicopter.

1. void init()

Used to call all the glut functions

1. void keys()

Used to initialise the w key for upward movement and m key for downward movement.

1. void main()

Enters the processing loop, never to return. Registered call back functions will be called when the corresponding events instigate them.

1. void mouse()

This function is used to set the left mouse button for upward movement of helicopter else it will go down.

1. void mouseheliD()

It calculates the mouse distance to be covered downwards

1. void mouseheliU()

It calculates the mouse distance to be covered upwards.

# **Chapter 4**

# **DESIGN**

**4.1 Initialization**

Initialize the interaction with the windows .Initialize the display mode-double buffer and depth buffer. Initialize the various call back functions for drawing and redrawing, for mouse and key board interfaces. Calculating distance travelled by copter and displaying and updating it continuously as score of player while game is on.

**4.2 Flow of control**

The flow of control in the below flow chart is respected to the Texture Package. For any of the program flow chart is compulsory to understand the program. We consider the flowchart for the texture project in which the flow starts from start and proceeds to the main function after which it comes to the initialization of call-back functions and further it proceeds to mouse and keyboard functions input and do the required operation. Finally when the game ends the running window will close and result window will open, end of flowchart.

**4.2ProjectDesign**

START

MAIN

INITIALIZE CALLBACK FUNCTIONS

MAINSCREENDISPLAYED

EXECUTE THE INPUT

PRINT RESULT

STOP

KEYBOARD

MOUSEINTERUPT

READING INPUT

END OF GAME

# **Chapter 5**

# **IMPLEMENTATION**

**5.1 Built-in functions**

1. **glRasterPos3f( x, y, -1)**

Specifies the raster position for pixel operations.

1. **glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, st[i])**

Renders a bitmap charater using OpenGL from the specified array of characters, and in the specified font style.

1. **glutPostRedisplay()**

Marks the current window as needing to be redisplayed.

1. **glutTimerFunc( 100, TimeEvent, 1)**

Registers a timer callback to be triggered in a specified number of milliseconds.

1. **glClearColor (0.0, 0.0, 0.0, 0.0)**

Specifies clear values for the color buffers.

1. **glShadeModel (GL\_SMOOTH)**

Select flat or smooth shading. Specifies a symbolic value representing a shading technique. Accepted values are GL\_FLAT and GL\_SMOOTH.

1. **glEnable(GL\_DEPTH\_TEST)**

Enables the OpenGL capabilities, Specifies the conditions under which the pixels will be drawn.

1. **glLightfv(GL\_LIGHT1 ,GL\_AMBIENT, LightAmbient)**

Returns light source parameter values.

1. **gluQuadricDrawStyle( Cylinder, GLU\_FILL)**

Specifies the draw style required for quadrics.

1. **glPushMatrix()**and**glPopMatrix()**

Push and pop the current matrix stack.

1. **glTranslatef() and glRotatef()**

Multiplies current matrix by Translation and Rotation matrix respectively.

1. **glMatrixMode (GL\_PROJECTION)**

Specifies which matrix is the current matrix.

1. **glLoadIdentity()**

Replaces current matrix with identity matrix.

1. **gluLookAt()**

Defines a viewing transformation.

1. **glutSwapBuffers()**

Swaps the buffers of the current window if double buffered.

1. **glViewport()**

Sets the viewport.

1. **glutInitDisplayMode (GLUT\_DOUBLE | GLUT\_RGB)**

Sets the initial display mode.

1. **glutInitWindowSize (500, 500)**and **glutInitWindowPosition (50, 50)**

Set the initial window size and position respectively.

1. **glutCreateWindow()**

Creates a top level window with the window name as specified.

1. **glutAddMenuEntry()**

 Adds a menu entry to the bottom of the current menu.

1. **glutAttachMenu(GLUT\_RIGHT\_BUTTON)**

Attaches a mouse button for the current window to the identifier of the current menu**.**

1. **glutDisplayFunc(display)**

Sets the display callback for the current window.

1. **glutReshapeFunc(reshape)**

Sets the reshape callback for the current window.

1. **glutMainLoop()**

Enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

**5.2 Source Code**

#include<stdlib.h>

#include<GL/glut.h>

#include<time.h>

#include<stdio.h>

#include<conio.h>

#include<windows.h>

float bspd=0.02; // block dx value

char name[25];

float b1x=50.0,b1y=0;//block 1 init position

float hm=0.0;//copter moving dy value

int i=0,sci=1;float scf=1; // for increment score score\_intscore\_flag

char scs[20],slevel[20];

//to store score\_string using itoa() and level as well

int level=1,lflag=1,wflag=1; //level\_flag&welcome\_flaginit w/ 1

void init(void)

{

srand(time(0));

b1y=(rand()%45)+10;//b/w 10 to 44

glClearColor (0.0, 0.0, 0.0, 0.0);

glShadeModel (GL\_SMOOTH);

glLoadIdentity ();

glOrtho(0.0, 100.0, 0.0, 100.0, -1.0 , .0);

}

void drawcopter()

{

glColor3f(0.7,1.0,1.0);

glRectf(10,49.8,19.8,44.8);//body

glRectf(2,46,10,48);//tail

glRectf(2,46,4,51);//tail up

glRectf(14,49.8,15.8,52.2);//propeller stand

glRectf(7,53.6,22.8,52.2);//propeller\*/

}

void renderBitmapString(float x,float y,float z,void \*font,char\*string)

{

char \*c;

glRasterPos3f(x, y,z);

for(c=string; \*c != '\0'; c++)

{

glutBitmapCharacter(font, \*c);

}

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

//GameOver Checking

if(

(i==730||i==-700)

//top and bottom checking

||

( ((int)b1x==10||(int)b1x==7||(int)b1x==4||(int)b1x==1) &&(int)b1y<53+(int)hm&&(int)b1y+35>53+(int)hm)

// propeller front checking

||

( ((int)b1x==9||(int)b1x==3||(int)b1x==6) &&(int)b1y<45+(int)hm&&(int)b1y+35>45+(int)hm)

//lower body checking

||

( ((int)b1x==0) && (int)b1y<46+(int)hm&&(int)b1y+35>46+(int)hm))

// lower tail checking

{

glColor3f(0.0,0.0,1.0);

glRectf(0.0,0.0,100.0,100.0);

glColor3f(1.0,0.0,0.0);

renderBitmapString(40,70,0,GLUT\_BITMAP\_HELVETICA\_18,"GAME OVER!!!");

glColor3f(1.0,1.0,1.0);

renderBitmapString(25,58,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24,"You");

renderBitmapString(45,58,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24,"scored:");

renderBitmapString(70,58,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24,scs);

glutSwapBuffers();

glFlush();

printf("\nGAME OVER\n\n");

printf("%s\You scored %s" ,name,scs);

printf("\n\nClose the console window to exit...\n");

//getch();

exit(0);

}

else if(wflag==1)//Welcome Screen

{

wflag=0;

glColor3f(0.0,0.5,0.7);

glRectf(0.0,0.0,100.0,10.0);//ceil

glRectf(0.0,100.0,100.0,90.0);//floor

glColor3f(1.0,1.0,1.0);

renderBitmapString(35,85,0,GLUT\_BITMAP\_HELVETICA\_18,"CITY ENGINEERING COLLEGE");

renderBitmapString(41,80,0,GLUT\_BITMAP\_HELVETICA\_12,"Bangalore, Karnataka-560 062");

glColor3f(1.0,1.0,0.0);

renderBitmapString(20,65,0,GLUT\_BITMAP\_8\_BY\_13,"a mini project for Computer Graphics & Visualization Laboratery");

renderBitmapString(45.5,70,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24,"Helicopter");

glColor3f(1.0,0.0,0.0);

renderBitmapString(40,45,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24,"Welcome");

renderBitmapString(53,45,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24,name);

renderBitmapString(43,30,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24,"Click To Start");

renderBitmapString(17,24,0,GLUT\_BITMAP\_9\_BY\_15,"CLICK AND HOLD LEFT MOUSE BUTTON TO GO UP RELEASE TO GO DOWN");

glColor3f(0.0,0.0,0.0);

drawcopter();

glutSwapBuffers();

glFlush();

}

else

{

//on every increase by 50 in score in each level

if(sci%50==0&&lflag==1)

{

lflag=0; //make level\_flag=0

level++;//increase level by 1

bspd+=0.01;//increase block\_dx\_speed by 0.01

}

//within every level make level\_flag=1

else if(sci%50!=0&&lflag!=1)

{

lflag=1;

}

glPushMatrix();

glColor3f(0.0,0.5,0.7);

glRectf(0.0,0.0,100.0,10.0); //ceil

glRectf(0.0,100.0,100.0,90.0); //floor

glColor3f(0.0,0.0,0.0); //score

renderBitmapString(1,3,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24,"Distance:");

//glColor3f(0.7,0.7,0.7);

sprintf(slevel,"%d",level); //level

renderBitmapString(80,3,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24,"Level:");

renderBitmapString(93,3,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24,slevel);

scf+=0.025; //so less as program run very fast

sci=(int)scf;

sprintf(scs,"%d",sci);

//from int to char convertion to display score

renderBitmapString(20,3,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24,scs);

glTranslatef(0.0,hm,0.0);

// hm(=dy) changes occur by mouse func

drawcopter();

//code for helicopter

//if wall move towards left & get out of projection volume

if(b1x<-10)

{

b1x=50; //total width is 50

b1y=(rand()%25)+20;

//10 for selling+10 for giving enough space

// block bottom limit 0+20 & top limit 24+20=44

}

else

b1x-=bspd;

//within the projection volume dec its x value by block\_speed

glTranslatef(b1x,-hm,0.0);

glColor3f(1.0,0.0,0.0);

glRectf(b1x,b1y,b1x+5,b1y+35);//block 1

glPopMatrix();

glutSwapBuffers();

glFlush();

}

}

void moveHeliU(void)

{

hm+=0.05;

i++;

glutPostRedisplay();

}

void moveHeliD()

{

hm-=0.05;

i--;

glutPostRedisplay();

}

void mouse(int button, int state, int x, int y)

{

switch (button)

{

case GLUT\_LEFT\_BUTTON:

if (state == GLUT\_DOWN)

glutIdleFunc(moveHeliU);

else if (state == GLUT\_UP)

glutIdleFunc(moveHeliD);

break;

default: break;

}

}

void keys(unsigned char key,intx,int y)

{

if(key=='w') glutIdleFunc(moveHeliU);

if(key=='m') glutIdleFunc(moveHeliD);

}

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

{

printf("enter your name to play: ");

scanf("%s",name);

glutInit(&argc, argv);

glutInitDisplayMode (GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize (800, 600);

glutInitWindowPosition (200,200);

glutCreateWindow ("2D Copter Game");

init();

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutKeyboardFunc(keys);

glutMainLoop();

return 0;

}

**Chapter 6**

**TESTING**

Testing in general means validation and verification. It shows that the system conforms to its specifications and system meets all expectation of the user.

**6.1 Test case for mouse and keyboard:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl.No. | Test Case  Description | Expected Result | Actual Result | Remarks |
| 1.  2. | Click the LEFT mouse button on the display screen.  Click the keyboard options:   * w * m | Helicopter will move upwards else move downwards.  Helicopter will move in upward direction.  Helicopter will move in downward direction. | Helicopter will move upwards else move downwards.  Helicopter will move in upward direction.  Helicopter will move in downward direction. | Pass  Pass  Pass |

**Chapter 7**

**SNAPSHOTS**

* 1. **Welcome Window Screen**

****

**Figure 6.1:** welcome window screen

* 1. **Game Running Window**

**Figure 6.2:** game running window

**7.3 Game over**

You scored : 50

**Chapter 8**

**CONCLUSION**

We have attempted to design and implement “Helicopter Game”. OpenGl supports enormous flexibility in the design and the use of OpenGl graphics programs. The presence of many built in classes methods take care of much functionality and reduce the job of coding as well as makes the implementation simpler.

The project was started with the designing phase in which we figured the requirements needed, the layout design, then comes the detail designing of each function after which, was the testing and debugging stage. We 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. Computer Graphics – Principals And Practice (Foley, Van Dam, Fenier and Hughes) helped me to understand graphics generation algorithms, user interface and dialogue design

2. OpenGL Programming Guide (Addison-Wesley Publishing Company) helped me to get through all OpenGL functions and Commands and understandings of all aspects of them.

3. www.cplusplus.com: - provided references regarding all c++ functions and their uses.

4. www.stackoverflow.com: - help to get rid of all types of error occurred regarding uses of OpenGL functions.

5. www.lighthouse3d.com: - OpenGL tutorial for implementing the OpenGL functions in Source code.