**Chapter 1**

**INTRODUCTION TO COMPUTER GRAPHICS**

**1.1.1 Computer Graphics**

Computer graphics are graphics created using computer and more general, the representation and manipulation of pictorial data by a computer.

The development of computer graphics has made computers easier to interact with and better for understanding and interpreting many types of data. Development in computer graphics had a profound impact on many types of media and has revolutionized the animation and video game industry.

The term Computer Graphics includes almost everything on computers that is not text or sound. Today nearly all computers use some graphics and users expect to control their computer through icons and pictures rather than just by typing. The term Computer Graphics has several meanings:

* The representation and manipulation of pictorial data by a computer.
* The various technologies used to create and manipulate such pictorial data.
* The sub field of computer science which studies methods for digitally synthesizing and manipulating visual contents see study of computer science.

Today computers and computer-generated images touch many aspects of our daily life. Computer imagery is found on television, in newspaper, in weather report, and during surgical procedures. A well-constructed can present complex statistics in a form that is easier to understand and interpret. Such graphs are used to illustrate papers, reports, and other presentation materials. A range of tools and facilities are available to enable users to visualize their data, and computer graphics are used in many disciplines.

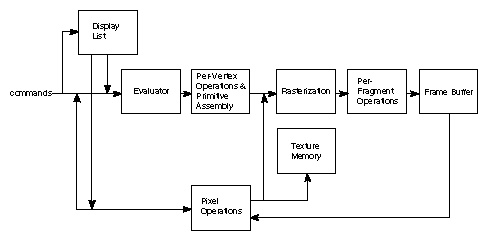
**1.1.2 Applications of Computer Graphics**

* Computational biology
* Computational physics
* Computer-aided design
* Computer simulation
* Digital art
* Education
* Graphic design
* Video Games
* Virtual reality
* Web design

**1.2 Introduction to OpenGL**

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

**1.2.1 Basic OpenGL Operation**

****

**Fig 1.2.1 The OpenGLBlock Diagram**

The figure 1.2.1 shown above gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

As shown by the first block in the diagram, rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a *display list* for processing at a later time.

The evaluator stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. During the next stage, per*-*vertex operations and primitive assembly, OpenGL processes geometric primitives—points, line segments, and polygons, all of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for the next stage.

The evaluator stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. During the next stage, per-vertex operations and primitive assembly, OpenGL processes geometric primitives—points, line segments, and polygons, all of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for the next stage.

Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each fragmentso produced is fed into the last stage, per-fragment operations, which performs the final operations on the data before it's stored as pixels in the frame buffer. These operations include conditional updates to the frame buffer based on incoming and previously stored z-values (for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

Input data can be in the form of pixels rather than vertices. Such data, which might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the pixel operations stage. The result of this stage is either stored as texture memory, for use in the rasterization stage, or rasterized and the resulting fragments merged into the frame buffer just as if they were generated from geometric data. All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

**1.2.2 GLUT**

GLUT (pronounced like the glutin gluttony) is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works on both Win32 PCs and X11 workstations.

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 like Motif. GLUT is simple, easy, and small. My intent is to keep GLUT that way.

The GLUT library supports the following functionality:

* Multiple windows for OpenGL rendering.
* Callback driven event processing.
* An `idle' routine and timers.
* Utility routines to generate various solid and wire frame objects.
* Support for bitmap and stroke fonts.
* Miscellaneous window management functions.

**Chapter 2**

**FUNCTIONS USED**

**2.1 Built-in functions**

**glColor3f (float red, float green, float blue)**

This function is used to mention the color in which the pixel should appear.The number 3 specifies the number of arguments that the function would take . ‘f’ gives the type that is float. The arguments in the order RGB (Red, Green, Blue). The color of the pixel can be specified as the combination of 3 primary colors.

**glClearColor(int red,int green, int blue, int alpha);**

This function is used to clear the color of the screen. The 4 values that are passed as arguments for this function are (Red, Green, Blue, Alpha) where the red, green and blue components are taken to set the background color and alpha is the value that specifies depth of the window. It is used for 3D images.

**void roof (GLint rux, GLint ruy, GLint rdx,GLint rdy)**

This function is used to create the roof of the 3 huts.

**glFlush ();**

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. GlFlush empties all of thesebuffer, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

**void glutInit(int argc,char \*\*argv)**

glutInit will initialize the glut library and negotiate a session with the window system. During this process, glutInit may cause the termination of the glut program with an error message to the user if the glut cannot be properly initialized. Examples of this initialization include the failure to connect to the window system, the lack of window system to support OpenGL, and invalid command line options. glutInit also processes command line options, but the specific options parse are window system dependent.

**void glutReshapeFunc(void(\*Func)(int width, int height))**

glutReshapeFunc sets the reshape callback for the current window.The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window’s first display callback after the window is created or whenever an overlay for the window is established. The width and height parameters of the callback specify the new window size in pixels. Before the callback, the current window is set to the window that has been reshaped.

If a reshape callback is not registered for a window or NULL is passed to glutReshapeFunc, the default reshape callback is used.

**void glutMainLoop(void)**

glutMainLoop enters the glut event processing loop. This routine should be called atmost once in the glut program.

**glutInitDisplayMode(unsigned int mode);**

Requests a display with the properties in mode. The value of mode is determined by the logical OR.

**glutInit(int \*argc,char \*\*argv);**

Initializes Glut.

**void glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z)**

glRotated and glRotatef functions multiply the current matrix by rotation matrix.

**void glTranslatef(TYPE x,TYPE y, TYPE z)**

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

**int glutCreateMenu(void(\*Func)(int value));**

Returns an identifier for a top-level menu and registers the callback function ‘f’ that returns an integer value corresponding to the menu entry selected. glutCreateMenu creates a new popup menu and returns a unique small integer identifier.

**void glutAddMenuEntry(char \*name, int value)**

This function adds an entry with the string name displayed to the current menu. Value is returned to the menu callback when the entry is selected.

**void glutAttachMenu(int button)**

glutAttachMenu attaches a mouse button for the current window to the identifier of the current menu.

**void visibility(int status)**

This is the function, where all the function are called back.This is where all the gluts are initialized . All the animations work happens through the callback register.

**void glOrtho(GLdoubleleft, GLdoubleright, GLdoubletop, GLdouble bottom,GLdoublenear,GLdoublefar)**

Defines an orthographic viewing volume with all parameters measured from the center of projection plane.

**glutDisplayFunc(void(\*Func)(void));**

Whenever glut determines the contents of the window need to be redisplayed, the callback function registered by glutDisplayFunc() is executed.

**void glutKeyboardFunc(void(\*Func)(unsigned char key,int x,int y))**

Registers the keyboard callback function. The callback function returns the ASCII code of the key pressed and the position of the mouse.

**void glutMouseFunc(void (\*Func)(int button, int state,int x, int y))**

Registers the mouse callback function.The callback function returns the button(GLUT\_LEFT\_BUTTON,GLUT\_RIGHT\_BUTTON,GLUT\_MIDDLE\_BUTTON),the state of the button after the event(GLUT\_UP,GLUT\_DOWN) and the position of the mouse with respect to the top left corner of the window.

**Chapter 3**

**FUNCTION MODULES**

**Main function**

This main function is capable of handling the arguments given in the argument list at the command prompt as we have used variable ‘argc’ for total number of arguments and ‘argv’ for the array of argument list. Main function initializes the Display Mode, Window Size and position. Then it invokes the display function within glutDisplayFunc() as the call back function.

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

{

int c\_menu;

printf("--------------------------------------------------------------------------------");

printf(" Simple Village ");

printf("--------------------------------------------------------------------------------\n\n");

printf("Press 'd' or 'D' to make it day. \n\n");

printf("Press 'n' or 'N' to make it night. \n\n");

printf("Press 'b' or 'B' to fly Birds. \n\n");

printf("Press 'l' or 'L' to turn On the lights. \n\n");

printf("Press 'f' or 'F' to turn Off the lights. \n\n");

printf("Press RIGHT MOUSE BUTTON to display menu. \n\n");

printf("Press LEFT MOUSE BUTTON to quit the program. \n\n\n");

printf("Press any key and Hit ENTER.\n");

scanf("%s",&ch);

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(1100.0,700.0);

glutInitWindowPosition(0,0);

glutCreateWindow("Simple Village");

glutDisplayFunc(display);

glutIdleFunc(idle);

glutKeyboardFunc(keyboardFunc);

glutMouseFunc(mouse);

myinit();

c\_menu=glutCreateMenu(main\_menu);

glutAddMenuEntry("Aeroplane",1);

glutAddMenuEntry("Comet",2);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

return 0;

}

**KeyBoard Function**

This Interactive function is invoked at the main program within glutKeyboardFunc() and it is repeatedly called during the execution of the program and hence handles the keyboard interrupts from the users. When a key is been pressed the ASCII code of the key and the position of the screen pointer at the time of interruption will be sent to the function.

KeyBoard Function is used as an interface between the User Defined Women Function as shown below and the Wind Energy Source Code which is used to show as an Additional Modification in a Main program.

void keyboardFunc( unsigned char key, int x, int y )

{

switch( key )

{

case 'd':

case 'D':

day=1;

p=0.75;

q=0.47;

r=0.14;

break;

case 'n':

case 'N':

day=0;

p=0.52;

q=0.37;

r=0.26;

break;

case 'b':

case 'B':

bird=1;

i=800;

b=0.0;

count=0;

break;

case 'l':

case 'L':

e=0.90;

f=0.91;

g=0.98;

break;

case 'f':

case 'F':

e=0.0;

f=0.0;

g=0.0;

break;

};

**Menu Function**

This Interactive function is invoked at the main program within glutCreateMenu() and it is

repeatedly called during the execution of the program and hence handles the Mouse interrupts

from the users.

When a button is been pressed the identity of the button and the state of the button and the

position of the screen pointer at the time of interruption will be sent to the menu function.

void main\_menu(int index)

{

switch(index)

{

case 1:

if(index==1)

{

plane=1;

o=n=0.0;

}

break;

case 2:

if(index==2)

{

comet=1;

c=0.0;

}

break;

}

}

**Display functions**

We have used three display functions in this program the first one that is the myDisplay function is called in the main program using glutDisplayFunc() call back function. The display function controls the displaying of the front screen, help screen or the display function.

The display function that is display() calls the objects like house(),Clouds() etc. According to the order that we have written hence it indirectly handles the depth information.

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

draw\_object();

glFlush();

}

**BackGround Function**

This function is used to give a background touch to the concept being developed.

ex. The color of the sky,clouds,ground which is represented as blue,white and green when the program is being executed.

void background()

{

glColor3f(0.0,0.1,0.0);

glBegin(GL\_POLYGON);//green ground

glVertex2i(-250.0,-250.0);

glVertex2i(250.0,-250.0);

glVertex2i(250.0,0.0);

glVertex2i(-250.0,0.0);

glEnd();

glColor3f(0.1 ,0.1,0.1);

glBegin(GL\_POLYGON);//mid night blue sky

glVertex2i(-250.0,0.0);

glVertex2i(-250.0,250.0);

glVertex2i(250.0,250.0);

glVertex2i(250.0,0.0);

glEnd();

}

**Fan Function**

This function indicates the fan-like object which is used to rotate placed above the tip of fanpole in order to generate the electrical energy.The fan object is made to rotate at its fixed point using Translate and Rotate functions.The code for this is as given below.

void fan()

{

glPushMatrix();

glLoadIdentity();

glColor3f(1,1,1);

glTranslatef (-8.0,20.0, 2.0);/\*rotation about fixed point\*/

glRotatef(spin,0.0,0.0,1.0);

glTranslatef (8.0,-20.0, -2.0);

glBegin(GL\_TRIANGLES);/\*for all 4 fans\*/

glVertex3f(-8.0,20.0,2.0);

glVertex3f(-12.0,16.0,3.0);

glVertex3f(-12.0,18.0,4.0);

glVertex3f(-8.0,20.0,2.0);

glVertex3f(-4.0,24.0,3.0);

glVertex3f(-4.0,22.0,4.0);

glEnd();

glPopMatrix();

}

**Wire Function**

This function is defined in order to represent connections between wind mills that are being developed and the powerstation,which along supplies electric energy to various houses and streetlights indicating the generation of electricity. Simple attribute functions along with polygonal lines are used.The code developed is as given below.

void wires()

{

glColor3f(.7,.5,.7);

glEnable(GL\_LINE\_STIPPLE);

glLineStipple(1,0x00FF);

glBegin(GL\_LINES);

glVertex2f(-8.0,7.0);

glVertex2f(-32.0,7.0);

glVertex2f(-8.0,10.0);

glVertex2f(1.5,10.0);

glVertex2f(26.5,7.0);

glVertex2f(14.5,12.0);

glVertex2f(31.0,7.0);

glVertex2f(18.0,12.0);

glEnd();

glDisable(GL\_LINE\_STIPPLE);

}

**Powerstation Function**

This function is defined in order to represent the storage and controlling medium for the electricity being generated through wind mills from the above defined model.

the powerstation in the project represents the supply of electricity to various house of the area under wind mills.

Attribute functions along with various input vertices and polygon functions are used during development of this function. The code for execution is given as below.

void powerstation()

{

GLint ax=1.5,ay=8;

glColor3f(1.0,0.25,0.1);

glBegin(GL\_POLYGON);//from tip(anti clkwise)

glVertex2i(ax,ay);//a

glVertex2i (ax-2,ay-2);//b

glVertex2i(ax-2,ay-8);//c

glVertex2i(ax+2,ay-8);//d

glVertex2i(ax+2,ay-2);//e

glEnd();

glColor3f(0.7,0.5,0.3);

glBegin(GL\_POLYGON);//roof (from a)

glVertex2i(ax,ay+3);//a

glVertex2i(ax-3,ay-3);//b

glVertex2i(ax+3,ay-3);//e

glEnd();

glColor3f(v,v,w);

glBegin(GL\_POLYGON);/\* door \*/

glVertex2i(ax-1,ay-5.0);//top left

glVertex2i(ax-1.0,ay-8.0);//bottom left

glVertex2i(ax+1.0,ay-8.0);// bottom right

glVertex2i(ax+1.0,ay-5.0);//top right

glEnd();

}

**Grass Function**

Grass function is use to create gree grass picture in the village the grass are created around the house..

grass

glColor3f(0.6,0.8,0.196078);

glBegin(GL\_POLYGON);

glVertex2f(0,160);

glVertex2f(0,380);

glVertex2f(1100,380);

glVertex2f(1100,160);

glEnd();

}

**Cloud Function**

It is one of the important functionthat is being developed for the project,

indicating that the movement of clouds in the sky medium leading to rotation of wind mills in clockwise or anti-clockwise directions due to flow of wind in the sky medium.

The clouds that are being developed are made to move using various transformation and translate functions. The code is given as below.

cloud1

for(l=0;l<=20;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(160+m,625,l);

}

for(l=0;l<=35;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(200+m,625,l);

draw\_circle(225+m,625,l);

}

for(l=0;l<=20;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(265+m,625,l);

}

cloud2

for(l=0;l<=20;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(370+m,615,l);

}

for(l=0;l<=35;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(410+m,615,l);

draw\_circle(435+m,615,l);

draw\_circle(470+m,615,l);

}

for(l=0;l<=20;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(500+m,615,l);

}

**House Function**

The roof and hut functions that are being defined represents the house-like objects on the output window that is being developed. House function consisting of window() ,door(), chim() ,ect...

chim

glColor3f(0.35,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(540,330);

glVertex2f(540,430);

glVertex2f(960,430);

glVertex2f(960,330);

glEnd();

home

glColor3f(p,q,r);

glBegin(GL\_POLYGON);

glVertex2f(550,100);

glVertex2f(550,330);

glVertex2f(950,330);

glVertex2f(950,100);

glVertex2f(850,100);

glVertex2f(850,250);

glVertex2f(650,250);

glVertex2f(650,100);

glEnd();

window border

glColor3f(0.35,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(595,205);

glVertex2f(595,285);

glVertex2f(675,285);

glVertex2f(675,205);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(825,205);

glVertex2f(825,285);

glVertex2f(905,285);

glVertex2f(905,205);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(845,205);

glVertex2f(845,285);

glVertex2f(850,285);

glVertex2f(850,205);

glEnd();

door

glColor3f(e,f,g);

glBegin(GL\_POLYGON);

glVertex2f(800,100);

glVertex2f(800,220);

glVertex2f(700,220);

glVertex2f(700,100);

glEnd();

glColor3f(0.35,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(760,120);

glVertex2f(760,200);

glVertex2f(700,220);

glVertex2f(700,100);

glEnd();

window

glColor3f(e,f,g);

glBegin(GL\_POLYGON);

glVertex2f(600,210);

glVertex2f(600,280);

glVertex2f(670,280);

glVertex2f(670,210);

glEnd();

**Tree Function**

Tree function is used in this project to create a village environment ,tree contributr to their environment by providing oxygen ,improving air quality,climate ect….

tree

glColor3f(0.9,0.2,0.0);

glBegin(GL\_POLYGON);

glVertex2f(280,185);

glVertex2f(280,255);

glVertex2f(295,255);

glVertex2f(295,185);

glEnd();

for(l=0;l<=30;l++)

{

glColor3f(0.0,0.5,0.0);

draw\_circle(270,250,l);

draw\_circle(310,250,l);

}

for(l=0;l<=25;l++)

{

glColor3f(0.0,0.5,0.0);

draw\_circle(280,290,l);

draw\_circle(300,290,l);

}

for(l=0;l<=20;l++)

{

glColor3f(0.0,0.5,0.0);

draw\_circle(290,315,l);

}

//tree 1

glColor3f(0.9,0.2,0.0);

glBegin(GL\_POLYGON);

glVertex2f(100,135);

glVertex2f(100,285);

glVertex2f(140,285);

glVertex2f(140,135);

glEnd();

for(l=0;l<=40;l++)

{

glColor3f(0.0,0.5,0.0);

draw\_circle(40,280,l);

draw\_circle(90,280,l);

draw\_circle(150,280,l);

draw\_circle(210,280,l);

draw\_circle(65,340,l);

draw\_circle(115,340,l);

draw\_circle(175,340,l);

}

for(l=0;l<=55;l++)

{

glColor3f(0.0,0.5,0.0);

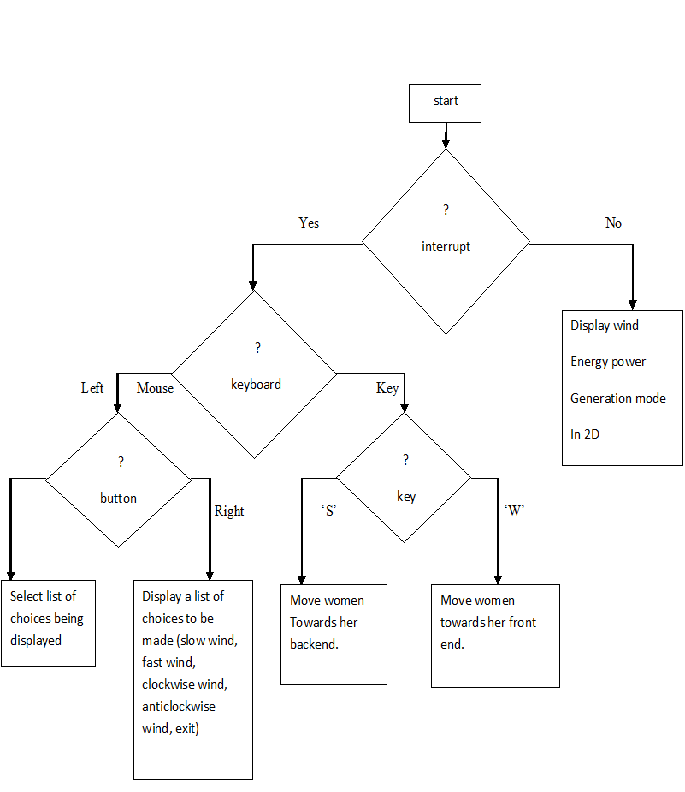
draw\_circle(115,360,l);

}

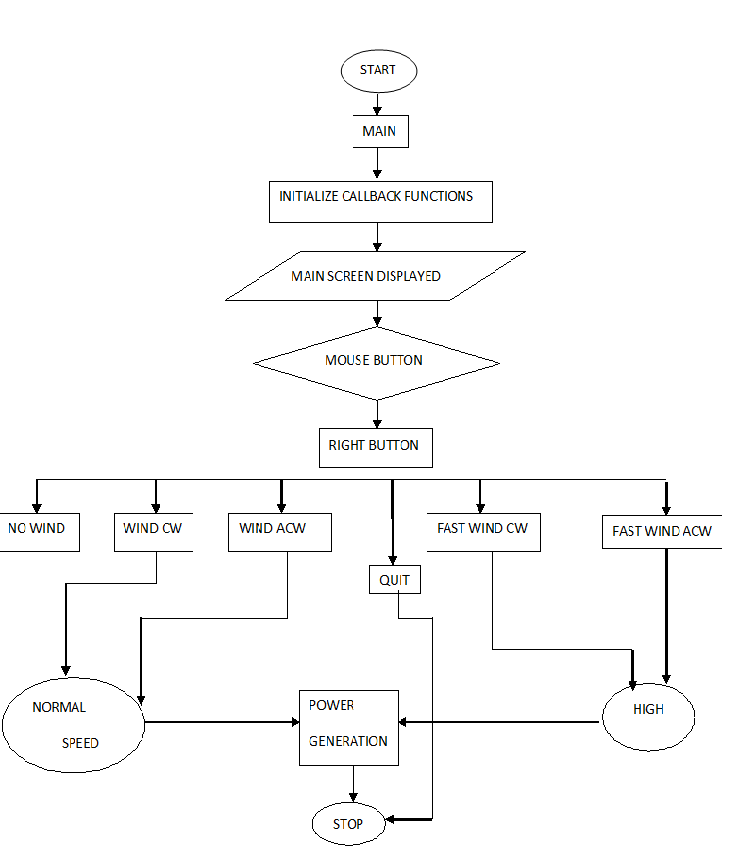
**Chapter 4**

**SYSTEM DESIGN**

**4.1 Architectural Diagram**



**4.2 Flowchart**



**Chapter 6**

**IMPLEMENTATION**

#include<stdio.h>

#include<GL/glut.h>

#include <GL/gl.h>

#include <stdlib.h>

#define SPEED 30.0

float i=0.0,m=0.0,n=0.0,o=0.0,c=0.0,b=0.0;

float p=0.75,q=0.47,r=0.14;

float e=0.90,f=0.91,g=0.98;

int count=0;

int light=1,day=1,plane=0,comet=0,xm=900,bird=0;

char ch;

void declare(char \*string)

{

while(\*string)

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*string++);

}

void draw\_pixel(GLint cx, GLint cy)

{

glBegin(GL\_POINTS);

glVertex2i(cx,cy);

glEnd();

}

void plotpixels(GLint h,GLint k, GLint x,GLint y)

{

draw\_pixel(x+h,y+k);

draw\_pixel(-x+h,y+k);

draw\_pixel(x+h,-y+k);

draw\_pixel(-x+h,-y+k);

draw\_pixel(y+h,x+k);

draw\_pixel(-y+h,x+k);

draw\_pixel(y+h,-x+k);

draw\_pixel(-y+h,-x+k);

}

void draw\_circle(GLint h, GLint k, GLint r)

{

GLint d=1-r, x=0, y=r;

while(y>x)

{

plotpixels(h,k,x,y);

if(d<0) d+=2\*x+3;

else

{

d+=2\*(x-y)+5;

--y;

}

++x;

}

plotpixels(h,k,x,y);

}

void draw\_object()

{

int l;

if(day==1)

{

//sky

glColor3f(0.0,0.9,0.9);

glBegin(GL\_POLYGON);

glVertex2f(0,380);

glVertex2f(0,700);

glVertex2f(1100,700);

glVertex2f(1100,380);

glEnd();

//sun

for(l=0;l<=35;l++)

{

glColor3f(1.0,0.9,0.0);

draw\_circle(100,625,l);

}

//plane

if(plane==1)

{

glColor3f(1.0,1.0,1.0);

glBegin(GL\_POLYGON);

glVertex2f(925+n,625+o);

glVertex2f(950+n,640+o);

glVertex2f(1015+n,640+o);

glVertex2f(1030+n,650+o);

glVertex2f(1050+n,650+o);

glVertex2f(1010+n,625+o);

glEnd();

glColor3f(0.8,0.8,0.8);

glBegin(GL\_LINE\_LOOP);

glVertex2f(925+n,625+o);

glVertex2f(950+n,640+o);

glVertex2f(1015+n,640+o);

glVertex2f(1030+n,650+o);

glVertex2f(1050+n,650+o);

glVertex2f(1010+n,625+o);

glEnd();

}

//cloud1

for(l=0;l<=20;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(160+m,625,l);

}

for(l=0;l<=35;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(200+m,625,l);

draw\_circle(225+m,625,l);

}

for(l=0;l<=20;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(265+m,625,l);

}

//cloud2

for(l=0;l<=20;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(370+m,615,l);

}

for(l=0;l<=35;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(410+m,615,l);

draw\_circle(435+m,615,l);

draw\_circle(470+m,615,l);

}

for(l=0;l<=20;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(500+m,615,l);

}

//grass

glColor3f(0.6,0.8,0.196078);

glBegin(GL\_POLYGON);

glVertex2f(0,160);

glVertex2f(0,380);

glVertex2f(1100,380);

glVertex2f(1100,160);

glEnd();

}

else

{

//sky

glColor3f(0.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(0,380);

glVertex2f(0,700);

glVertex2f(1100,700);

glVertex2f(1100,380);

glEnd();

//moon

int l;

for(l=0;l<=35;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(100,625,l);

}

//star1

glColor3f(1.0,1.0,1.0);

glBegin(GL\_TRIANGLES);

glVertex2f(575,653);

glVertex2f(570,645);

glVertex2f(580,645);

glVertex2f(575,642);

glVertex2f(570,650);

glVertex2f(580,650);

glEnd();

//star2

glColor3f(1.0,1.0,1.0);

glBegin(GL\_TRIANGLES);

glVertex2f(975,643);

glVertex2f(970,635);

glVertex2f(980,635);

glVertex2f(975,632);

glVertex2f(970,640);

glVertex2f(980,640);

glEnd();

//star3

glColor3f(1.0,1.0,1.0);

glBegin(GL\_TRIANGLES);

glVertex2f(875,543);

glVertex2f(870,535);

glVertex2f(880,535);

glVertex2f(875,532);

glVertex2f(870,540);

glVertex2f(880,540);

glEnd();

//star4

glColor3f(1.0,1.0,1.0);

glBegin(GL\_TRIANGLES);

glVertex2f(375,598);

glVertex2f(370,590);

glVertex2f(380,590);

glVertex2f(375,587);

glVertex2f(370,595);

glVertex2f(380,595);

glEnd();

//star5

glColor3f(1.0,1.0,1.0);

glBegin(GL\_TRIANGLES);

glVertex2f(750,628);

glVertex2f(745,620);

glVertex2f(755,620);

glVertex2f(750,618);

glVertex2f(745,625);

glVertex2f(755,625);

glEnd();

//star6

glColor3f(1.0,1.0,1.0);

glBegin(GL\_TRIANGLES);

glVertex2f(200,628);

glVertex2f(195,620);

glVertex2f(205,620);

glVertex2f(200,618);

glVertex2f(195,625);

glVertex2f(205,625);

glEnd();

//star7

glColor3f(1.0,1.0,1.0);

glBegin(GL\_TRIANGLES);

glVertex2f(100,528);

glVertex2f(95,520);

glVertex2f(105,520);

glVertex2f(100,518);

glVertex2f(95,525);

glVertex2f(105,525);

glEnd();

//star8

glColor3f(1.0,1.0,1.0);

glBegin(GL\_TRIANGLES);

glVertex2f(300,468);

glVertex2f(295,460);

glVertex2f(305,460);

glVertex2f(300,458);

glVertex2f(295,465);

glVertex2f(305,465);

glEnd();

//star9

glColor3f(1.0,1.0,1.0);

glBegin(GL\_TRIANGLES);

glVertex2f(500,543);

glVertex2f(495,535);

glVertex2f(505,535);

glVertex2f(500,532);

glVertex2f(495,540);

glVertex2f(505,540);

glEnd();

//comet

if(comet==1)

{

for(l=0;l<=7;l++)

{

glColor3f(1.0,1.0,1.0);

draw\_circle(300+c,675,l);

}

glColor3f(1.0,1.0,1.0);

glBegin(GL\_TRIANGLES);

glVertex2f(200+c,675);

glVertex2f(300+c,682);

glVertex2f(300+c,668);

glEnd();

}

//Plane

if(plane==1)

{

for(l=0;l<=1;l++)

{

glColor3f(1.0,0.0,0.0);

draw\_circle(950+n,625+o,l);

glColor3f(1.0,1.0,0.0);

draw\_circle(954+n,623+o,l);

}

}

//grass

glColor3f(0.0,0.3,0.0);

glBegin(GL\_POLYGON);

glVertex2f(0,160);

glVertex2f(0,380);

glVertex2f(1100,380);

glVertex2f(1100,160);

glEnd();

}

//Ground

glColor3f(0.0,0.3,0.0);

glBegin(GL\_POLYGON);

glVertex2f(-600,0);

glVertex2f(-600,185);

glVertex2f(1100,185);

glVertex2f(1100,0);

glEnd();

//tree

glColor3f(0.9,0.2,0.0);

glBegin(GL\_POLYGON);

glVertex2f(280,185);

glVertex2f(280,255);

glVertex2f(295,255);

glVertex2f(295,185);

glEnd();

for(l=0;l<=30;l++)

{

glColor3f(0.0,0.5,0.0);

draw\_circle(270,250,l);

draw\_circle(310,250,l);

}

for(l=0;l<=25;l++)

{

glColor3f(0.0,0.5,0.0);

draw\_circle(280,290,l);

draw\_circle(300,290,l);

}

for(l=0;l<=20;l++)

{

glColor3f(0.0,0.5,0.0);

draw\_circle(290,315,l);

}

//tree 1

glColor3f(0.9,0.2,0.0);

glBegin(GL\_POLYGON);

glVertex2f(100,135);

glVertex2f(100,285);

glVertex2f(140,285);

glVertex2f(140,135);

glEnd();

for(l=0;l<=40;l++)

{

glColor3f(0.0,0.5,0.0);

draw\_circle(40,280,l);

draw\_circle(90,280,l);

draw\_circle(150,280,l);

draw\_circle(210,280,l);

draw\_circle(65,340,l);

draw\_circle(115,340,l);

draw\_circle(175,340,l);

}

for(l=0;l<=55;l++)

{

glColor3f(0.0,0.5,0.0);

draw\_circle(115,360,l);

}

//chim

glColor3f(0.35,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(540,330);

glVertex2f(540,430);

glVertex2f(960,430);

glVertex2f(960,330);

glEnd();

//home

glColor3f(p,q,r);

glBegin(GL\_POLYGON);

glVertex2f(550,100);

glVertex2f(550,330);

glVertex2f(950,330);

glVertex2f(950,100);

glVertex2f(850,100);

glVertex2f(850,250);

glVertex2f(650,250);

glVertex2f(650,100);

glEnd();

//window border

glColor3f(0.35,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(595,205);

glVertex2f(595,285);

glVertex2f(675,285);

glVertex2f(675,205);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(825,205);

glVertex2f(825,285);

glVertex2f(905,285);

glVertex2f(905,205);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(845,205);

glVertex2f(845,285);

glVertex2f(850,285);

glVertex2f(850,205);

glEnd();

//door

glColor3f(e,f,g);

glBegin(GL\_POLYGON);

glVertex2f(800,100);

glVertex2f(800,220);

glVertex2f(700,220);

glVertex2f(700,100);

glEnd();

glColor3f(0.35,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(760,120);

glVertex2f(760,200);

glVertex2f(700,220);

glVertex2f(700,100);

glEnd();

//window

glColor3f(e,f,g);

glBegin(GL\_POLYGON);

glVertex2f(600,210);

glVertex2f(600,280);

glVertex2f(670,280);

glVertex2f(670,210);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(830,210);

glVertex2f(830,280);

glVertex2f(900,280);

glVertex2f(900,210);

glEnd();

glColor3f(0.35,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(620,210);

glVertex2f(620,280);

glVertex2f(625,280);

glVertex2f(625,210);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(650,210);

glVertex2f(650,280);

glVertex2f(655,280);

glVertex2f(655,210);

glEnd();

glColor3f(0.35,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(850,205);

glVertex2f(850,285);

glVertex2f(855,285);

glVertex2f(855,205);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(880,205);

glVertex2f(880,285);

glVertex2f(885,285);

glVertex2f(885,205);

glEnd();

if(bird==1)

{

/\*glColor3f(0.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(300+i-xm,250+b);

glVertex2f(330+i-xm,250+b);

glVertex2f(330+i-xm,280+b);

glEnd();\*/

glColor3f(0.73,0.16,0.96 );

glBegin(GL\_POLYGON);

glVertex2f(300+i-xm,265+b);

glVertex2f(330+i-xm,265+b);

glVertex2f(330+i-xm,250+b);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(330+i-xm,275+b);

glVertex2f(340+i-xm,275+b);

glVertex2f(330+i-xm,265+b);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(200+i-xm,285+b);

glVertex2f(230+i-xm,285+b);

glVertex2f(230+i-xm,270+b);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(230+i-xm,295+b);

glVertex2f(240+i-xm,295+b);

glVertex2f(230+i-xm,285+b);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(150+i-xm,285+b);

glVertex2f(180+i-xm,285+b);

glVertex2f(180+i-xm,270+b);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(180+i-xm,295+b);

glVertex2f(190+i-xm,295+b);

glVertex2f(180+i-xm,285+b);

glEnd();

}

glFlush();

}

void idle()

{

if(light==0 && (i>=0 && i<=1150))

{

i+=SPEED/10;

m+=SPEED/150;

n-=2;

o+=0.2;

c+=2;

}

if(light==0 && (i>=2600 && i<=3000))

{

i+=SPEED/10;

m+=SPEED/150;

n-=2;

o+=0.2;

c+=2;

}

if(light==0)

{

i=i;

m+=SPEED/150;

n-=2;

o+=0.2;

c+=2;

}

if(count<=3)

{

glClearColor(1.0,1.0,1.0,1.0);

i+=SPEED/10;

b+=SPEED/10;

m+=SPEED/150;

n-=2;

o+=0.2;

c+=2;

}

if(i>1900)

i=800.0;

if(m>1100)

m=0.0;

if( o>75)

{

plane=0;

}

if(c>500)

{

comet=0;

}

if(b>500)

{

b=0.0;

i=800.0;

count=count+1;

}

glutPostRedisplay();

}

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

{

if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_UP)

exit(0);

}

void keyboardFunc( unsigned char key, int x, int y )

{

switch( key )

{

case 'd':

case 'D':

day=1;

p=0.75;

q=0.47;

r=0.14;

break;

case 'n':

case 'N':

day=0;

p=0.52;

q=0.37;

r=0.26;

break;

case 'b':

case 'B':

bird=1;

i=800;

b=0.0;

count=0;

break;

case 'l':

case 'L':

e=0.90;

f=0.91;

g=0.98;

break;

case 'f':

case 'F':

e=0.0;

f=0.0;

g=0.0;

break;

};

}

void main\_menu(int index)

{

switch(index)

{

case 1:

if(index==1)

{

plane=1;

o=n=0.0;

}

break;

case 2:

if(index==2)

{

comet=1;

c=0.0;

}

break;

}

}

void myinit()

{

glClearColor(1.0,1.0,1.0,1.0);

glColor3f(0.0,0.0,1.0);

glPointSize(2.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,1100.0,0.0,700.0);

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

draw\_object();

glFlush();

}

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

{

int c\_menu;

printf("--------------------------------------------------------------------------------");

printf(" Simple Village ");

printf("--------------------------------------------------------------------------------\n\n");

printf("Press 'd' or 'D' to make it day. \n\n");

printf("Press 'n' or 'N' to make it night. \n\n");

printf("Press 'b' or 'B' to fly Birds. \n\n");

printf("Press 'l' or 'L' to turn On the lights. \n\n");

printf("Press 'f' or 'F' to turn Off the lights. \n\n");

printf("Press RIGHT MOUSE BUTTON to display menu. \n\n");

printf("Press LEFT MOUSE BUTTON to quit the program. \n\n\n");

printf("Press any key and Hit ENTER.\n");

scanf("%s",&ch);

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(1100.0,700.0);

glutInitWindowPosition(0,0);

glutCreateWindow("Simple Village");

glutDisplayFunc(display);

glutIdleFunc(idle);

glutKeyboardFunc(keyboardFunc);

glutMouseFunc(mouse);

myinit();

c\_menu=glutCreateMenu(main\_menu);

glutAddMenuEntry("Aeroplane",1);

glutAddMenuEntry("Comet",2);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

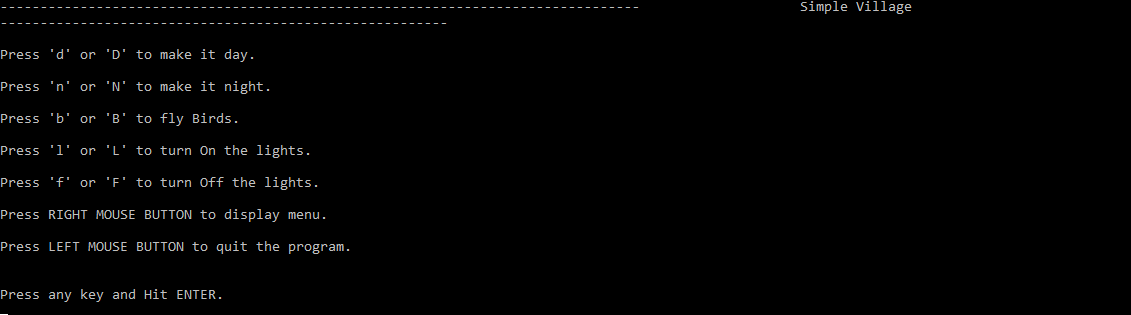
return 0;

}

**Chapter 7**

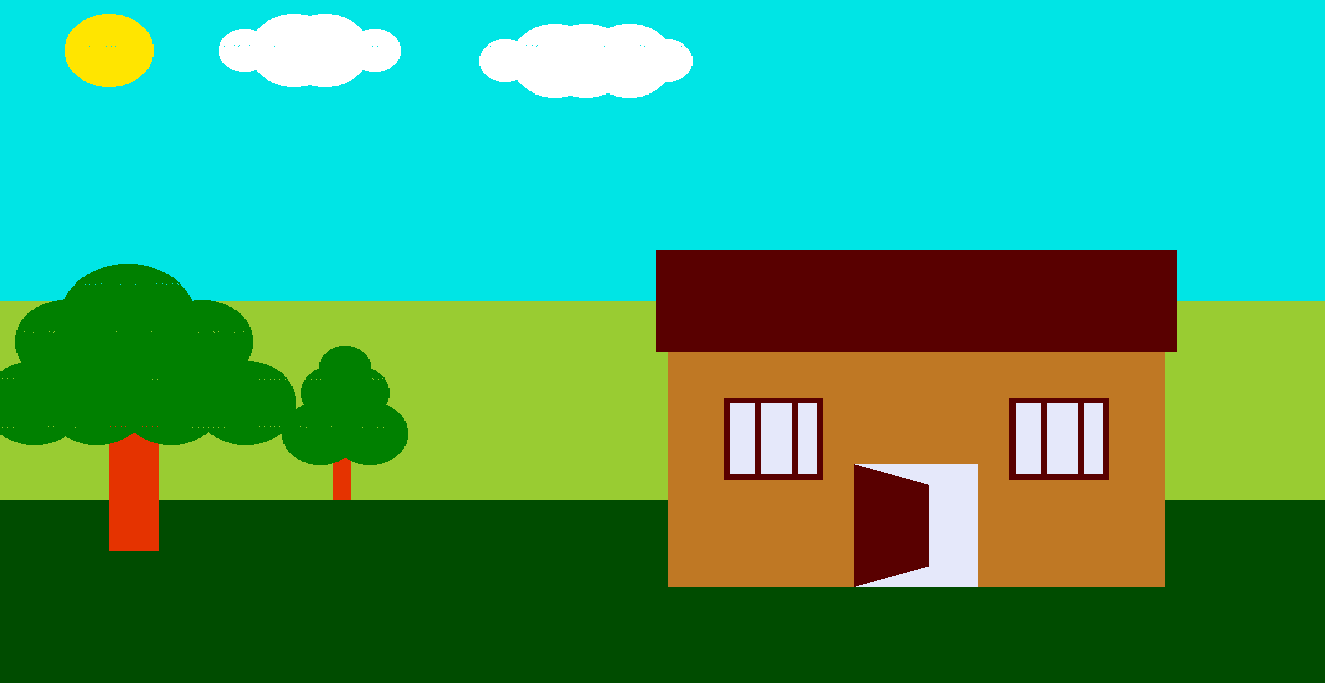
**SNAPSHOTS**

**Snapshot 1**



The above is the snapshot of the village ,its show the different option like d/D- make it day, n/N-for male it night, b/B-for flying the bird, l/L-for turn on the home lights.etc……

**Snapshot 2**

.

In this snapshots its shows the complete image of the village ,like trees ,sun, clouds, and home, grass, etc…….

**Snapshot 3**



This shot shows the night mode ,in this mode we can see the moon ,stars and ect.... main thing is light are turned on in the home while user going to press the l/L-turn of the lights …..

**Snapshot 4**



This shot shows the night mode ,in this mode we can see the moon ,stars and ect.... main thing is light are remains off in the home while user going to press the f/F-turn off the lights …..

**CHAPTER 8**

**CONCLUSION**

The illumination effect produced by a light source has been shown using Computer Graphics in ‘Microsoft visual C++’ and ‘OpenGL libraries.

**C**omputer graphics is concerned with all aspects of producing pictures or images using a computer, we use particular graphics software system called **OpenGL.** which has become a widely accepted standard for developing graphic application. Using OpenGL functions user can create geometrical objects and can use translation, rotation, with respect to the co-ordinate system

An attempt has been made to develop an OpenGL graphics package which meets all the necessary requirements that were set out. It was a whole new experience and fun to learn, itis user friendly and provides an easy interaction for the user. The user can very easily use this tool to draw or manipulate a drawing. The interface and keyboard driven and the user can select a function by pressing any key option representing that function. Finally, this graphics package satisfies all requirements and provides good entertainment.

**Chapter 9**

**References**

**BOOKS**

[1] Edward Angel Interactive Computer Graphics Pearson Education 5th Edition:

[2] OpenGL programming guide : Dave Shreiner Mason Woo

[3] OpenGL super bible : Richard S, Wright

[4] 3D Computer Graphics: A Mathematical Introduction with OpenGL, by S. Buss, Cambridge University Press, 2003.

**WEBSITES**

[1] www.OpenGL simple examples.

[2] www.OpenGL Redbook.

[3] www.OpenGL programming guide

[4] www.glut3.7