***COMPUTER GRAPHICS AND VISUALIZATION LABORATORY (6CSL04)***

**2018-2019(EVEN)**

##### AN OPEN-ENDED PROJECT REPORT ON

**“AQUARIUM SIMULATOR”**

##### Submitted By:

**PANKEJ JAISWAL 1SI16CS133**



## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMKUR-572103

(An Autonomous Institute, Affiliated to Visvesvaraya Technological University, Belgaum, Recognized by AICTE, New Delhi)

**1. INTRODUCTION**

* 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.



* 1. **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 interfaces**

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



**2. LITERATURE SURVEY**

The basic functions like glcolor3f(…); gltotatef(..),gltranslate(..) etc that are most commonly used in the code are taken from the prescribed VTU Text book “INTERACTIVE COMPUTER GRAPHICS” 5thedition by Edward Angel.[1].

The lab programs in the syllabus also serve as a basic template for creating a project. The usage of colours 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 “object-oriented programming” by Sourav Sahay. [2].

Some concepts like constructing bowl and fountain are taken from the search results in codecolony.com.

**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 ***DEMOLITION OF A BUILDING BY AEROPLANE CRASH***

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:**

 Code blocks 17

 GNU GCC compiler & OpenGL library

**Hardware Requirements:**

 Processor-Intel or AMD(Advanced Micro Devices)

 RAM-512MB(minimum)

 Hard Disk-1MB(minimum)

 Mouse

 Keyboard

 Monitor

**4. DESIGN**

**4.1 User Defined Functions**

 **myinit():**

This 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\_BIT);**

**glFlush();**

 **timerfunc():**

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():**

Save 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.

**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:

ValueMeaning

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 parametersmeasured 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 CALL 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));

**5.4MAIN 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);

**#include <GL/glut.h>**

**#include <stdlib.h>**

**#include <math.h>**

**#include <unistd.h>**

**#include <iostream>**

**#define DEG2RAD 3.14159 / 180.0**

**using namespace std;**

**float x[10],buby[10]= {-.95},bubx=-3.14,r[10],offset[10],pos[10],z[10],yy[10],speed[10],sizee[10],grass[100],ff=-.6,tv[10]= {0};**

**int dx[10]= {1},bx=1,zx[10],ee[10],fx=1;**

**int colors[10][3] = {{255, 165, 0},**

**{242, 38, 19},**

**{44, 62, 80},**

**{219, 10, 91},**

**{236,236,236},**

**{243, 156, 18},**

**{102, 51, 153},**

**{224, 130, 131},**

**{207, 0, 15},**

**{162, 222, 208}**

**};**

**void circle(float x, float y, float r);**

**void DrawEllipse(float x, float y, float radiusX, float radiusY);**

**void DrawBubble(int i,float t);**

**void DrawGrass(int i,float t);**

**void DrawFish(int, int\*,float);**

**void Drawcircle(float x, float y, float r,float n,bool f1,bool f2);**

**void ground();**

**void display()**

**{**

**glClear(GL\_COLOR\_BUFFER\_BIT); // Clear the color buffer**

**glClearColor(66/255.,161/255.,198/255.,1); // Black and opaque**

**glEnable(GL\_BLEND);**

**glEnable(GL\_POLYGON\_SMOOTH);**

**glLineWidth(2);**

**// ground**

**ground();**

**//fish**

**for(int i=0; i<5; i++)**

**DrawFish(i,colors[i],sizee[i]);**

**//grass**

**for(int i=0; i<3; i++)**

**DrawGrass(i,-.95+0.05\*i);**

**//bubble**

**for(int i=0; i<10; i++)**

**DrawBubble(i,pos[i]);**

**glutPostRedisplay(); // Post a re-paint request to activate display()**

**glutSwapBuffers(); // Double buffered - swap the front and back buffers**

**}**

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

**{**

**srand(time(0));**

**for(int i=0; i<10; i++)**

**{**

**r[i] = (float)(rand()%4+4)/1000;**

**offset[i] = (2\*(float)rand()/RAND\_MAX)-1;**

**pos[i]=2\*(float)rand()/RAND\_MAX-1;**

**yy[i] = (rand()%(9- (-7)+1)+-7)/10.0;**

**sizee[i] = (float)(rand()%5)/10+.3;**

**speed[i] = (float)(rand()%70)/10000+.001;**

**z[i]=0;**

**zx[i]=1;**

**ee[i] = -10+2\*i;//2\*(rand()%2)-10; //-10,-8,-6**

**x[i] = (2\*(float)rand()/RAND\_MAX)-1;**

**tv[i]=0;**

**}**

**ee[0]=-9;**

**ee[1]=-10;**

**ee[2]=-9;**

**for(int i=0; i<100; i++)**

**if (rand()%10<1)**

**grass[i] = rand()%10;**

**else**

**grass[i]=0;**

**glutInit(&argc, argv); // Initialize GLUT**

**glutInitDisplayMode(GLUT\_DOUBLE); // Enable double buffered mode**

**glutInitWindowSize(640, 640); // Set the window's initial width & height - non-square**

**glutInitWindowPosition(50, 50); // Position the window's initial top-left corner**

**glutCreateWindow("Fish"); // Create window with the given title**

**glutDisplayFunc(display); // Register callback handler for window re-paint event**

**glutMainLoop();**

**}**

**void ground()**

**{**

**glPushMatrix();**

**glTranslatef(0,-.5,0);**

**glRotatef(50,1,0,0);**

**glPolygonMode(GL\_FRONT\_AND\_BACK,GL\_FILL);**

**glBegin(GL\_POLYGON);**

**glColor3f(66/255.,161/255.,198/255.);**

**glVertex2d(-1, -0.25);**

**glColor3ub(230, 126, 34);**

**glVertex2d(-1, -1);**

**glVertex2d(1, -1);**

**glColor3f(66/255.,161/255.,198/255.);**

**glVertex2d(1, -0.25);**

**glEnd();**

**glPopMatrix();**

**}**

**void DrawFish(int i, int c0[],float sc)**

**{**

**//body**

**glPushMatrix();**

**glColor3ub(c0[0],c0[1],c0[2]);**

**if(dx[i]==1)**

**{**

**glScalef(1,1,1);**

**DrawEllipse(x[i], 0+yy[i], .18\*sc, .12\*sc);**

**}**

**else**

**{**

**glScalef(-1,1,1);**

**DrawEllipse(-x[i], 0+yy[i], .18\*sc, .12\*sc);**

**}**

**glPopMatrix();**

**//lower**

**glPushMatrix();**

**glTranslatef(0,+yy[i],0);**

**if(dx[i]==1)**

**{**

**glScalef(1,1,1);**

**glRotatef(40, 1, 0, 1);**

**glTranslatef(x[i],0,x[i]);**

**}**

**else**

**{**

**glScalef(-1,1,1);**

**glRotatef(40, 1, 0, 1);**

**glTranslatef(-x[i],0,-x[i]);**

**}**

**DrawEllipse(-.17\*sc, .035\*sc, .1\*sc, .075\*sc);**

**glPopMatrix();**

**// upper**

**glPushMatrix();**

**glTranslatef(0,+yy[i],0);**

**if(dx[i]==1)**

**{**

**glScalef(1,1,1);**

**glRotatef(-40, 1, 0, 1);**

**glTranslatef(x[i],0,x[i]);**

**}**

**else**

**{**

**glScalef(-1,1,1);**

**glRotatef(-40, 1, 0, 1);**

**glTranslatef(-x[i],0,-x[i]);**

**}**

**DrawEllipse(-.17\*sc, - .035\*sc, .1\*sc, .075\*sc);**

**glPopMatrix();**

**// eye**

**glPushMatrix();**

**glColor3f(1,1,1);**

**if(dx[i]==1)**

**{**

**glScalef(1,1,1);**

**DrawEllipse(x[i]+.1\*sc,0+.03\*sc+yy[i],.025\*sc,.025\*sc);**

**glColor3f(0,0,0);**

**DrawEllipse(x[i]+.1\*sc,0+.03\*sc+yy[i],.01\*sc,.01\*sc);**

**}**

**else**

**{**

**glScalef(-1,1,1);**

**DrawEllipse(-x[i]+.1\*sc,.03\*sc+yy[i],.025\*sc,.025\*sc);**

**glColor3f(0,0,0);**

**DrawEllipse(-x[i]+.1\*sc,.03\*sc+yy[i],.01\*sc,.01\*sc);**

**}**

**glPopMatrix();**

**dx[i] = x[i]>.87?-1:x[i]<-.87?1:dx[i];**

**x[i] = dx[i]==1?x[i]+speed[i]:x[i]-speed[i];**

**}**

**void circle(float x, float y, float r)**

**{**

**float t;**

**for (int i = -1000; i < 1000; i++)**

**{**

**t = i / 1000.0;**

**glVertex2f(t + x, sqrt(r \* r - t \* t) + y);**

**glVertex2f(t + x, -sqrt(r \* r - t \* t) + y);**

**}**

**}**

**void DrawEllipse(float x, float y, float radiusX, float radiusY)**

**{**

**register int i;**

**register float rad ;**

**glBegin(GL\_POLYGON);**

**for (i = 0; i < 360; i++)**

**{**

**glVertex2f(x + cos(i \* DEG2RAD) \* radiusX, y + sin(i \* DEG2RAD) \* radiusY);**

**}**

**glEnd();**

**}**

**void DrawBubble(int i,float t)**

**{// bubble**

**glPushMatrix();**

**if (i%2==0)**

**{**

**glColor4f(1,1,1,.3);**

**DrawEllipse(t+sin(bubx)/20,buby[i]+offset[i],.03,.03);**

**glColor3f(1,1,1);**

**glPointSize(1);**

**glBegin(GL\_POINTS);**

**circle(t+sin(bubx)/20,buby[i]+offset[i],.03);**

**glEnd();**

**}**

**else**

**{**

**glColor4f(1,1,1,.3);**

**DrawEllipse(t+cos(bubx)/20,buby[i]+offset[i],.03,.03);**

**glColor3f(1,1,1);**

**glPointSize(1);**

**glBegin(GL\_POINTS);**

**circle(t+cos(bubx)/20,buby[i]+offset[i],.03);**

**glEnd();**

**}**

**glPopMatrix();**

**buby[i]+=r[i];**

**if(buby[i]>1.3)**

**{**

**buby[i] =-1.5;**

**pos[i]=2\*(float)rand()/RAND\_MAX-1;**

**}**

**bubx +=0.01;**

**}**

**void DrawGrass(int k,float t)**

**{**

**glPushMatrix();**

**glLineWidth(6);**

**glBegin(GL\_LINE\_STRIP);**

**float j=0;**

**for(int i=0; i<79+ee[k]; j+=0.006,i++,tv[k]+=0.1) //(float)rand()/RAND\_MAX**

**{**

**glColor3f(109/255.0-j,154/255.0+j,109/255.0-j);**

**glVertex2f(t+sin(.9\*tv[k])/70 + z[k]\*cos(.005\*tv[k])/70,(float)i/150-1);**

**}**

**glEnd();**

**glPopMatrix();**

**z[k] += 0.001\*zx[k];**

**zx[k] = z[k]>1.56?-1:z[k]<-1.56?1:zx[k];**

**}**

**void Drawcircle(float x, float y, float r,float n=1000,bool f1=false,bool f2=false)**

**{**

**float t;**

**glBegin(GL\_POINTS);**

**for (int i = -n; i < n; i++)**

**{**

**t = i / 1000.0;**

**if (f1)**

**glVertex2f(t + x, sqrt(r \* r - t \* t) + y);**

**if(f2)**

**glVertex2f(t + x, -sqrt(r \* r - t \* t) + y);**

**}**

**glEnd();**

**}**

**8. 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 down approach by using Open GL by EDWARD ANGEL