Chapter 1

INTRODUCTION

Computer graphics are pictures and films created using computers. Usually, the term refers to computer-generated image data created with the help of specialized graphical hardware and software. It is a vast and recently developed area of computer science. The phrase was coined in 1960, by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, though sometimes erroneously referred to as computer-generated imagery (CGI).Some topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others. Computer graphics is responsible for displaying art and image data effectively and meaningfully to the consumer.

A “bouncing” ball is one of the simplest physics simulations yet provides a novice graphics programmer with a host of useful experience. The student creates a ball that bounces off the, then gamifies the whole experience into a simple game. It is designed to be accessible to students who have taken an introductory programming course and who have physics and algebra to the level of a high-school graduate.

1.1 What is OpenGL (Open Graphics Library):

OpenGL has become a widely accepted standard for developing graphics application. OpenGL is easy to learn, and it possesses most of the characteristics of other popular graphics system. It is top-down approach. OpenGL is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives.

Most of our applications will be designed to access openGL directly through functions in three libraries. Function in the main GL library have name that begin with the letter gl and stored in the library. The second is the openGL utility Library (GLU). This library uses only GL function but contains codes for creating common object and viewing. Rather than using a different library for each system we used available library called openGL utility toolkit (GLUT). It used as #include<glut.h>

A graphics editor is a computer program that allows users to compose and edit pictures interactively on the computer screen and save them in one of many popular “bitmap” or “raster” a format such as TIFF, JPEG, PNG and GIF.

Graphics Editors can normally be classified as:

- 2D Graphics Editors.
- 3D Graphics Editors

My project named “FLOWING FOUNTAIN MODEL” uses OpenGL software interface and develops 2D images. This project uses the techniques like Translation, motion, display list, transformation techniques, etc.

1.2What is GLUT?

GLUT is a complete API written by Mark Kilgard which lets you create windows and handle the messages. It exists for several platforms, that means that a program which uses GLUT can be compiled on many platforms without (or at least with very few) changes in the code.

1.3 How does OpenGL work?

OpenGL bases on the state variables. There are many values, for example the color, that remain after being specified. That means, you can specify a color once and draw several polygons, lines or whatever with this color then. There are no classes like in DirectX. However, it is logically structured. Before we come to the commands themselves, here is another thing.

To be hardware independent, OpenGL provides its own data types. They all begin with "GL". For example, GLfloat, GLint and so on. There are also many symbolic constants, they all begin with "GL_", like GL_POINTS, GL_POLYGON. Finally the commands have the prefix "gl" like glVertex3f(). There is a utility library called GLU, here the prefixes are "GLU_" and "glu". GLUT commands begin with "glut", it is the same for every library. You want to know which libraries coexist with the ones called before? There are libraries for every system, Windows has the wgl*-Functions, Unix systems glx* and so on.

A very important thing is to know, that there are two important matrices, which affect the transformation from the 3d-world to the 2d-screen: The projection matrix and the modelview matrix. The projection matrix contains information, how a vertex – let's say a "point" in space – shall be mapped to the screen. This contains, whether the projection shall be isometric or from a perspective, how wide the field of view is and so on. Into the other matrix you put information, how the objects are moved, where the viewer is and so on.

1.4 How can I use GLUT?

GLUT provides some routines for the initialization and creating the window (or fullscreen mode, if you want to). Those functions are called first in a GLUT application:

In your first line you always write glutInit(&argc, argv). After this, you must tell GLUT, which display mode you want – single or double buffering, color index mode or RGB and so on. This is done by calling glutInitDisplayMode(). The symbolic constants are connected by a logical OR, so you could use glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE). In later tutorials we will use some more constants here.

After the initialization you call glCreateWindow() with the window name as parameter. Then you can (and should) pass some methods for certain events. The most important ones are "reshape" and "display". In reshape you need to (re)define the field of view and specify a new area in the window, where OpenGL is allowed to draw to.

Display should clear the so called color buffer – let's say this is the sheet of paper – and draw our objects.

You pass the methods by glut*Func(), for example glutDisplayFunc(). At the end of the main function you call glutMainLoop(). This function doesn't return, but calls the several functions passed by glut*Func.

1.5 OPENGL RENDERING PIPELINE

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. This ordering, as shown in Figure 1-2, is not a strict rule of how OpenGL is implemented but provides a reliable guide for predicting what OpenGL will do.

The following diagram shows the Henry Ford assembly line approach, which OpenGL takes to processing data. Geometric data (vertices, lines, and polygons) follow the path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are treated differently for part of the process. Both types of data undergo the same final steps (rasterization and per-fragment operations) before the final pixel data is written into the framebuffer

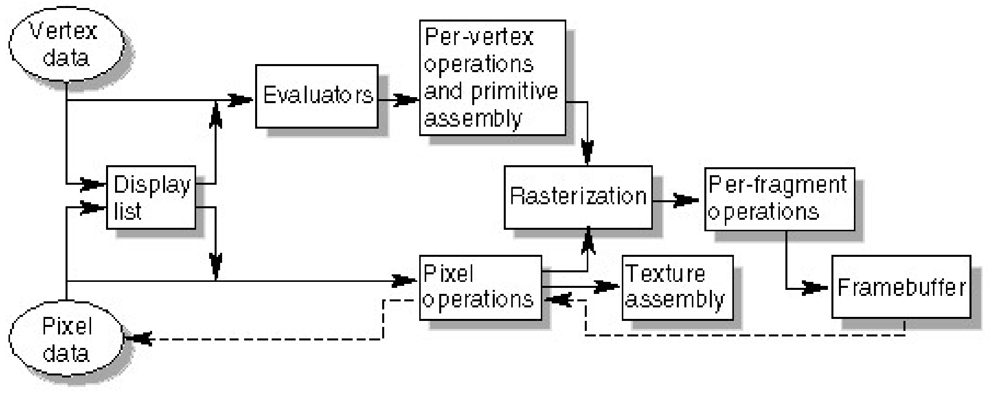


Fig 1.1: Order of Operations

Chapter 2

SYSTEM REQUIREMENTS

The minimum hardware and software requirements for this fun with polygons are specified below.

2.1 HARDWARE REQUIREMENTS

- Pentium or higher processor.
- 16 MB or more RAM.
- A standard keyboard, and Microsoft compatible mouse
- VGA monitor.
- If the user wants to save the Created files a secondary storage medium.

2.2 SOFTWARE REQUIREMENTS

• Operating system	:	Windows
• Software Used	:	Microsoft visual 8.0 using C++ (or) CodeBlocks
• Library Used	:	OPENGL Library

Chapter 3

Analysis and Design

3.1 Description

The main idea behind this project is to display the bouncing ball game with computer graphics. This graphics package is based on the OpenGL library functions. The programming language used here is C using OpenGL libraries.

In this project, we are demonstrating the game in which the ball will have a random motion and with every mouse click on the ball, the score will be incremented and the speed of the ball will be increased making the game difficult for the player. The ball will collide with the boundaries of the window and move in a random direction. The player needs to click on the ball, incrementing the score by 1 point for each consecutive click. The game has two levels. In Level 1, the player needs to score 10 points to proceed to the next level and next level, level 2 with increased difficulty.

The time allotted for each level is 10 seconds. The score will be displayed at the top for each level and time to keep a record of the points and time respectively. The final score is displayed to the player at the end of the game.

Chapter 4

IMPLEMENTATION

4.1 Steps

Step 1: [To create a Bouncing Ball]

The program starts with the initialization glutInit(&argc,agrv) and later Declare a class called CDrop. In GetNewPosition (), we calculate the position and delay of each drop with respect to the co-ordinate axes.

Step 2: [function createlist ()]

Dynamically allocate memory for the required vertices. Function glGenLists used to generate a contiguous set of empty display lists. Then specify a series of ‘for ‘loops to construct the top and bottom of the stone. Then create a qaudrilateral to represent the ground. To create water, use the following functions:

GlTranslatef () – is to calculate water and stone height. rand () function is used to generate random unique numbers, for each time it is executed.

Step 3: [function InitBouncing ()]

Declare StepAngle, the angle which the ray gets out of the fountain and RayAngle, the angle you see when you look down on the fountain. Use sine () and cosine () functions inside for loops, to calculate the speed of each step in the fountain, how many steps are required, that a drop comes out and falls down again.

Step 4: [Displaying]

[Keyboard function]

Manages operations by various keys pressed on the key board

[Display function]

Renders the program on to the screen.

Uses the following functions:

GlClear (GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)- Indicates the buffers currently enabled for color writing and also indicates the depth buffer.

GlPushMatrix (), glPopMatrix () — to push and pop the current matrix stack.

DrawTextXY () — used to set the text of the program.

glFlush () — force execution of GL commands in finite time.

GlutSwapBuffers ()-Swap the buffers ->make the result of rendering visible

[reshape function]

glMatrixMode (GL_PROJECTION) -applies subsequent matrix operations to the projection matrix stack.

glMatrix Mode (GL_MODELVIEW)-applies subsequent matrix operations to the model view matrix stack.

We adjust the viewing volume. We use the whole window for rendering and adjust point size to window size.

Step 5: [main function]

Here we specify the initial display mode, window size and position. Create a new window where the output is rendered. Create menus to move near, move away, move down, move up.

4.2 Sample Code

```
#include<GL/glut.h>
#include<math.h>
#include<stdbool.h>
#define PI 3.14159265f
#include<stdio.h>

GLfloat ballRadius = 0.2,xradius=0.2,xxradius=1.0,xxxradius=0.5;
GLfloat ballX = 0.0f;
GLfloat ballY = 0.0f;
GLfloat ballXMax,ballXMin,ballYMax,ballYMin;
GLfloat xSpeed = 0.02f;
GLfloat ySpeed = 0.007f;
int refreshMills = 30;
int refreshend = 60000;
```

```
GLfloat angle=0.0;
int xa,ya;
int flag=0,flag1=0,flag2=0,flagl=0,flagd=0;
int score = 0,score1=0,score2=0,endTimer,timer,i;
void *currentfont;
GLfloat  x, y;
GLdouble clipAreaXLeft,clipAreaXRight,clipAreaYBottom,clipAreaYTop;
void balldisp() ;
void scoredisp();
void end();

void drawstring(float x,float y,float z,char *string)
{
 char *c;
 glRasterPos3f(x,y,z);
 for(c=string;*c!='\0';c++)
  {
   glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,*c);
  }
}
void frontscreen(void)
{

                 glClear(GL_COLOR_BUFFER_BIT);
                glBegin(GL_TRIANGLE_FAN);
                for(angle=0; angle<360; angle+=1)
                {
                     y =(cos(angle*PI/180)*xradius);
                     x =(sin(angle*PI/180)*xradius);
                     glVertex2f(x,y);
                }

                glEnd();
 xradius=xradius+0.005;
```

```
  glColor3f( 1.0,1.0,0.0 );
 glBegin(GL_TRIANGLE_FAN);
                for(angle=0; angle<360.0; angle+=1)
                {
                     y =(sin(angle*PI/180)*xxradius);
                     x =(cos(angle*PI/180)*xxradius);
                     glVertex2f(x,y);
                }

                glEnd();
 xxradius=xxradius-0.005;
 glColor3f(0.0,0.0,0.0);
 drawstring(-0.42,0.9,0.0,"CGV MINI PROJECT");
glColor3f( 0.5,0.0,1.0 );
 drawstring(-0.4,0.4,0.0,"BOUNCING BALL");
 glColor3f( 0.0,0.0,0.0 );
 drawstring(-0.9,-0.0,0.0,"BY:");
 glColor3f( 0.0,0.0,0.0 );
 drawstring(-0.9,-0.1,0.0,"Yeswanth C");
drawstring(-0.9,-0.2,0.0,"Rohith R");
glColor3f( 0.0,0.0,1.5 );
 drawstring(-0.4,-0.7,0.0,"Press i for Instructions");
glColor3ub( rand()%250, rand()%250, rand()%200 );
 drawstring(-0.4,-0.9,0.0,"Press ENTER to Start");
 drawstring(-0.2,-0.55,0.0,"2020-21");
 glColor3f( 0.0,1.0,0.4 );

 glutSwapBuffers();
}

void level2()
{
                glClear(GL_COLOR_BUFFER_BIT);
                     drawstring(-0.175,0.0,0.0,"LEVEL 2 ");
```

```
                    drawstring(-0.4,-0.25,0.0,"Press c to Continue...");
                    glutSwapBuffers();
}

void finalsc()
{
                int z,j=0,k=0,y,a=0,b=0, w,d=0,e=0;
                z=score;
                y=score1;
                w=score2;
                glColor3f(1.0,1.0,0.0);
                glLoadIdentity();
   drawstring(-1.0,1.0,0.0,"FINAL SCORE : ");
                while(z > 9)
       {
          k = z % 10;

           glRasterPos2f (-0.20,1.0);
           glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,48+k);
                      z /= 10;
         glRasterPos2f(-0.25,1.0);
      }
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,48+z);
                drawstring(-1.0,0.9,0.0,"LEVEL 1 : ");
                while(w > 9)
       {
          e = w % 10;

           glRasterPos2f (-0.20,0.9);
           glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,48+e);
                      w /= 10;
         glRasterPos2f(-0.25,0.9);
      }
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,48+w);
                drawstring(-1.0,0.8,0.0,"LEVEL 2 : ");

                while(y > 9)
        {
           b = y % 10;

            glRasterPos2f (-0.20,0.8);
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,48+b);
                       y /= 10;
          glRasterPos2f(-0.25,0.8);
       }
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,48+y);
}

void end()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,1.0,0.0);
glBegin(GL_TRIANGLE_FAN);
                for(angle=0; angle<360; angle+=1)
                {
                     y =0.3+(cos(angle*PI/180)*xxxradius);
                     x =(sin(angle*PI/180)*xxxradius);
                     glVertex2f(x,y);
                }
 glEnd();
 glColor3f(0.0,0.0,0.0);
 glPointSize(15.0);
 glBegin(GL_POINTS);
 glVertex2f(-0.2,0.5);
 glVertex2f(0.2,0.5);
 glEnd();
 glLineWidth(10.0);
 glBegin(GL_LINES);
```

```
glVertex2f(0.0,0.2);
glVertex2f(0.0,0.4);
glEnd();
glBegin(GL_LINES);
glVertex2f(-0.2,0.0);
glVertex2f(0.0,0.1);
 glVertex2f(0.0,0.1);
  glVertex2f(0.2,0.0);
glEnd();
glColor3f(1.0,0.0,0.0);
drawstring(-0.3,-0.7,0.0,"GAME OVER");
finalsc();
glutSwapBuffers();

}
void instr()
{

                glClear(GL_COLOR_BUFFER_BIT);
                glColor3f( 0.5,0.0,1.0 );
                 drawstring(-0.25,0.85,0.0,"Instructions");
                  glColor3f(1.0,1.0,0.0);
                 drawstring(-0.9,0.55,0.0,"1. The Player needs to click on the ball in order");
                 drawstring(-0.8,0.45,0.0,"to score. ");
                 drawstring(-0.9,0.35,0.0,"2. Per click score will be incremented by 1. ");
                 drawstring(-0.9,0.25,0.0,"3. And the speed of the ball will increase ");
                 drawstring(-0.9,0.15,0.0,"4. To Quit at any stage press q ");
                 drawstring(-0.9,0.05,0.0,"5. Timer of 10 secs for two levels");
                 glColor3ub( rand()%250, rand()%250, rand()%200 );
    drawstring(-0.6,-0.9,0.0,"PRESS ENTER TO START");
                glutSwapBuffers();
}
```

```
void display()
{
                 glClear(GL_COLOR_BUFFER_BIT);
                glMatrixMode(GL_MODELVIEW);
                glLoadIdentity();
                balldisp();
                scoredisp();
                glutSwapBuffers();
                endTimer+=refreshMills;
                timer=endTimer/1000;
}
void mydisplay(void)
{   if(flag==0&&flag1==0&&flagd==0)
    frontscreen ();
    if(flag1==1&&flag==0)
        instr();
   if(flag==1&&flagd==0)
   {
   display();
                }
   if(score>=10&&flag1==0)
   {
   level2();
   endTimer=0;
   flagd=1;
   //level1sc=score;
}
     if(endTimer>=10000&&flag==1)
     {
   end();
   flagd=1;
}
   else if(endTimer>=20000&&flag1==1)
   {
```

```
  end();
        flagd=1;
}
}
void mykey( unsigned char key, int x, int y )
{
 switch(key)
 {

   case 13:  if(flag==0)
            flag=1;
           //flag1=0;
           //ySpeed=ySpeed+0.01;

           break;
   case 'i': if(flag==0)
                         flag1=1;
                       break;
   case 'c':if(flagl==0)
                      {
                     flag1=1;
                  flagd=0;

                    //score=0;
  ySpeed=ySpeed+0.05;}

                           break;
              case 'q' :
              case 'Q' : exit(0);
                      break;

 }

}
```

```
void balldisp()
{
            glTranslatef(ballX,ballY,0.0f);
            glBegin(GL_TRIANGLE_FAN);
            if(score>=10)
  {
  glColor3f(0,0,1);
  //level1sc=score;
  }
  else
  {

    glColor3f(1,0,0);
  }
            glVertex2f(0.0f,0.0f);
            int numSegments = 100;
            int i;
            for(i=0;i<=numSegments;i++)
            {
               angle = i*2.0f*PI/numSegments;
               glVertex2f(cos(angle)*ballRadius,sin(angle)*ballRadius);
            }

            glEnd();

            ballX += xSpeed;
            ballY += ySpeed;

            if(ballX > ballXMax)
            {   xa=ballX;
               ballX = ballXMax;
               xSpeed = -xSpeed;
            }
            else if(ballX < ballXMin)
            {   xa=ballX;
                ballX = ballXMin;
                xSpeed = -xSpeed;

            }
            if(ballY > ballYMax)
            {   ya=ballY;
                ballY = ballYMax;
                ySpeed = -ySpeed;

            }
            else if(ballY < ballYMin)
            {   ya=ballY;
                ballY = ballYMin;
                ySpeed = -ySpeed;
            }
}
void scoredisp()
{
            int z,j=0,k=0,i,s=0;
            z=score;
            glColor3f(1.0,1.0,0.0);
            glLoadIdentity();
            glRasterPos2f(0.28,1 );
            drawstring(0.38,1,0.0,"TIME : ");

            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,48+timer);
            if(score>=10)
    drawstring(-0.25,1,0,"LEVEL 2");
  else
    drawstring(-0.25,1,0,"LEVEL 1");
  drawstring(-1,1,0.0,"SCORE : ");
```

```
                while(z > 9)
        {
           k = z % 10;
            glRasterPos2f (-0.58,1);
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,48+k);
          z /= 10;
          glRasterPos2f(-0.62,1);
        }
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,48+z);
}

bool isMouseOverBall(float worldClickX, float worldClickY, float ballX, float ballY)
{
    float distance = sqrt(pow(worldClickX - ballX, 2.0f) + pow(worldClickY - ballY, 2.0f));
    return distance < ballRadius;
}

void windowToWorld(int windowX, int windowY, double *worldX, double *worldY)
{
    int x = windowX - 500 / 2;
    int y = windowY - 500 / 2;

    *worldX = (float)x / 250.0f;
    *worldY = -(float)y / 250.0f;
}

void onMouse(int button, int state, int x, int y)
{
                GLdouble worldClickX, worldClickY;
    windowToWorld(x, y, &worldClickX, &worldClickY);
    bool clicked = isMouseOverBall(ballX, ballY, worldClickX, worldClickY);
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
```

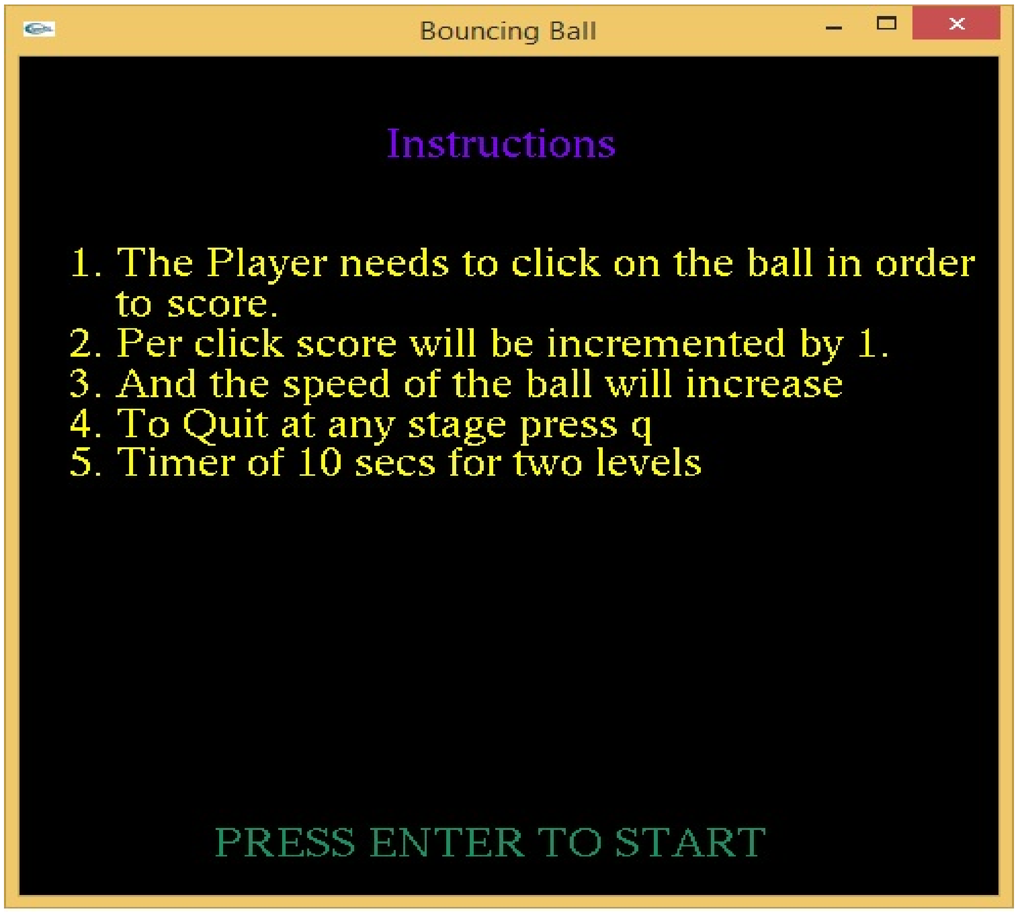
```
                 if(clicked)
   {
                score=score+1;

  if(score>10)
   {
                score1=score1+1;
                score2=10;
   }
   else
   {
                score2=score;
   }
  }
  }
}
void reshape(GLsizei width,GLsizei height)
{
                if(height ==0) height = 1;
                GLfloat aspect = (GLfloat)width / (GLfloat)height;
                glViewport(0,0,width,height);
                glMatrixMode(GL_PROJECTION);
                glLoadIdentity();
   if(width >=height)
                {
                     clipAreaXLeft = -1.0 * aspect;
                     clipAreaXRight = 1.0 * aspect;
                     clipAreaYBottom = -1.0;
                     clipAreaYTop = 1.0;
                }
                else
                {
                     clipAreaXLeft = -1.0;
                     clipAreaXRight = 1.0 ;
```

```
                    clipAreaYBottom = -1.0 / aspect;
                    clipAreaYTop = 1.0/ aspect;
               }
               gluOrtho2D(clipAreaXLeft,clipAreaXRight,clipAreaYBottom,clipAreaYTop+0.1
0);
               ballXMin = clipAreaXLeft + ballRadius;
               ballXMax =  clipAreaXRight - ballRadius;
               ballYMin =  clipAreaYBottom + ballRadius;
               ballYMax =  clipAreaYTop - ballRadius;
}

void Timer(int value)
{
               glutPostRedisplay();
               glutTimerFunc(refreshMills,Timer,5);
}

int main(int argc,char* argv[])
{
               glutInit(&argc,argv);
               glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
               glutInitWindowSize(500,500);
               glutInitWindowPosition(100,100);
               glutCreateWindow("Bouncing Ball");
               glutMouseFunc(onMouse);
   glutDisplayFunc(mydisplay);
               glutKeyboardFunc(mykey);
               glutReshapeFunc(reshape);
               glutTimerFunc(0,Timer,0);
               glutMainLoop();
}
```

Chapter 5

RESULTS

5.1 Instruction Page



Instructions

1. The Player needs to click on the ball in order to score.
2. Per click score will be incremented by 1.
3. And the speed of the ball will increase
4. To Quit at any stage press q
5. Timer of 10 secs for two levels

PRESS ENTER TO START

Fig. 5.1:Instruction page of bouncing ball:

5.2 Initial Page

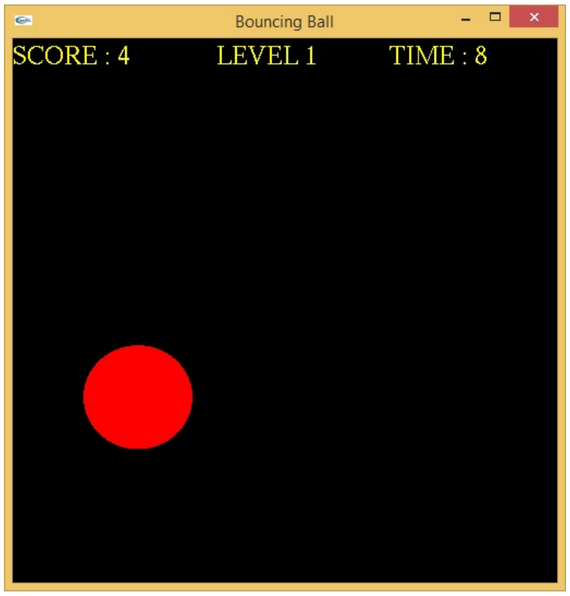
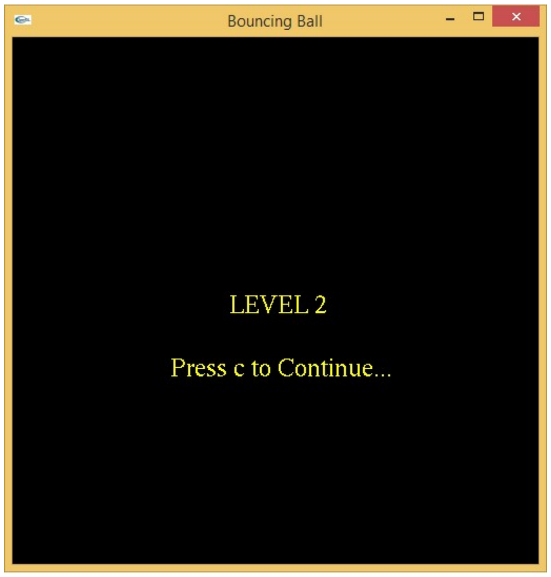


Fig.5.2: Initial Position of Bouncing Ball

5.3 Bouncing Ball Level



LEVEL 2

Press c to Continue...

Fig.5.3: Level of Bouncing Ball

5.4 Bouncing Ball Level

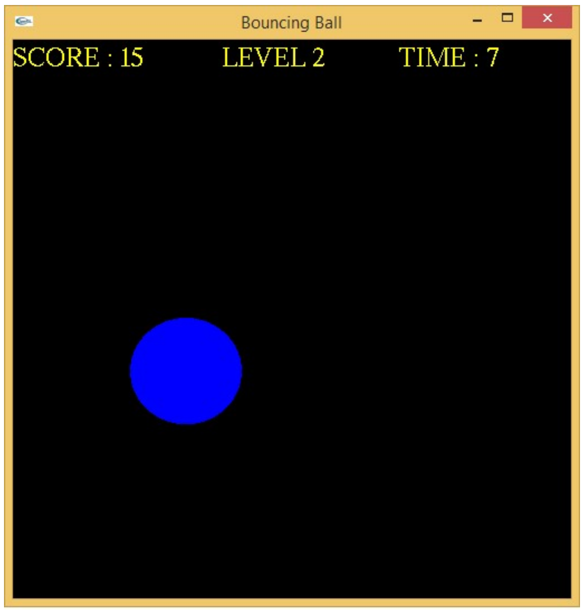


Fig.5.4:Level 2 starts of Bouncing Ball

5.5 Final Screen

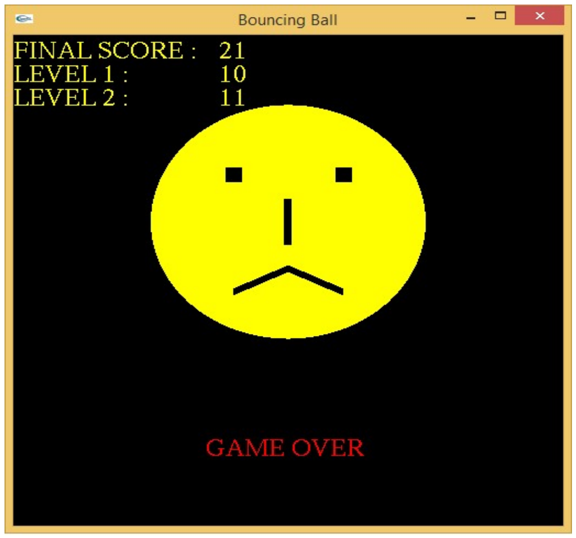


Figure 5.5: The final screen of Bouncing Ball:

Chapter 6

CONCLUSION

An attempt has been made to develop an OpenGL package which meets necessary requirements of the user successfully. Since it is user friendly, it enables the user to interact efficiently and easily.

The development of the mini project has given us a good exposure to OpenGL by which we have learnt some of the technique which help in development of animated pictures, gaming. Hence it is helpful for us even to take up this field as our career too and develop some other features in OpenGL and provide as a token of contribution to the graphics world.

REFERENCES

REFERENCES

BOOK REFERENCES

[1] - Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version, 3rd / 4th Edition, Pearson Education, 2011.

[2] - Edward Angel: Interactive Computer Graphics- a Top Down approach with OpenGL, 5th edition. Pearson Education, 2008.

WEBSITE REFERENCES

[1] - https://www.khronos.org/opengl/wiki/Getting_Started

[2] - https://www.opengl.org/sdk/docs/tutorials/OGLSamples/

[3] - https://www.geeksforgeeks.org/getting-started-with-opengl/

[4] - https://www.khronos.org/opengl/wiki//Code_Resources

[5] - http://www.lighthouse3d.com/tutorials/glut-tutorial/

[6] - http://code-blocks.blogspot.in/2014/12/bresenhams-circle-drawing-algorithm.html

[7] - https://github.com/sprintr/opengl-examples/blob/master/OpenGL-Menu.cpp