COMPUTER GRAPHICS AND VISUALIZATION

A Mini Project Report On

"FISHING MASTER"

Submitted in partial fulfilment of the Mini Project requirement for sixth semester of

Bachelor of Engineering
In
Computer Science & Engineering
Of
Visvesvaraya Technological University (VTU),
Belgaum During the year 2022-23

Submitted By

LAVANYA N (1SB20CS057)

Under the Guidance of

Prof. Sri Ramkumar R

Assistant professor

Dept. of CSE

Sri Sairam College of Engineering

Sai Leo Nagar, Guddanahalli, Anekal, Bengaluru – 562106

Department Of Computer Science and Engineering

CERTIFICATE

Certified that project work entitled "**FISHING MASTER**" is bonafide work carried out by

LAVANYA N (1SB20CS057)

In partial fulfilment for the award of the Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2022-23. It is certified that all corrections/suggestions indicated for the internal assessment have been incorporated in the report deposited in the department library. The project has been approved as it satisfies the academic requirements in respect of the project work prescribed for Bachelor of Engineering Degree.

Signature of Faculty Signature of HOD

External Viva

Name of the Examiners:

ACKNOWLEDGEMENT

It has been an honour and privilege to do my Seminar report on **FISHING MASTER.** I take this opportunity to convey my sincere thanks and regards to Our Chairman and Chief Executive Officer **SRI. SAI PRAKASH LEO MUTHU**, and **DR. ARUN KUMAR R**, Management Representative offered me a chance to study in this institute and everyone who has helped me in successful completion of this seminar.

I take immense pleasure in expressing my sincere gratitude to **Dr. B SHADAKSHARAPPA, PRINCIPAL, SRI SAIRAM COLLEGE OF ENGINEERING, BENGALURU** for giving me constant encouragement and support to achieve my goal.

My sincere thanks to **DR SMITHA J A, PROFESSOR & HEAD, DEPT. OF COMPUTER SCIENCE AND ENGINEERING** for permitting us to undertake the Seminar work and for his invaluable guidance.

I express my deepest thanks to our Mini Project Guide **PROF KARTHIKA P, ASSISTANT PROFESSOR**, **COMPUTER SCIENCE AND ENGINEERING** for taking part in useful decisions and guidance and arranging all facilities. I choose this moment to acknowledge her contribution gratefully.

I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and will continue to work on their improvement, in order to attain desired career objectives. Hope to continue cooperation with all of you in the future.

ABSTRACT

In this project we provide a whole new experience of fishing. The game visual consist of a fisher man with his boat in the water waiting for fresh fish. The main objective of this game is to catch the fish and each fish will add up ten points in the score.

The game has a time limit which is one minute within which the player have to max maximum score by fishing after the one minute the score of the player will be displayed .the player can catch the fish either by using the UP and DOWN arrow or using the left mouse button. If we press the UP or DOWN arrow key the string keeps on change the length which can allow the fish Fisherman to catch the fish .

To catch a fish in the FISHING MASTER game you must do two things, hook the fish by using the UP and DOWN arrow or by the left mouse button and when hooking a fish wait until its body touches the hook .once the hook touches the fish the user gain ten points .

The main menu gives three options

- Start game
- Help
- Quit

The start enables the player to start a new game under the time limit of one minute.

Help can show the controls.

And the quit exit to the desktop.

TABLE OF CONTENTS

Chapter no.

Chapter-1

INTRODUCTION

1.1 Computer Graphics

One of the most exciting and innovative fields in engineering today, computer graphics is an intricate combination of science, engineering, art, and psychology. This course introduces students to the technical concepts behind creating synthetic computer generated images, focusing on underlying mathematical concepts including triangles, normal, interpolation, texture mapping and bump mapping. Emphasis will be on developing practical skills for using graphics libraries and tools. Creativity and the production of impressive visual imagery are highly encouraged

The interaction and understanding of computers and interpretation of data has been made easier ,because of computer graphics. Computer graphic development has had a significant impact on many types of media and have revolutionized animation , movies and the video game industry.

Computer generated imagery can be categorized into several different types: two dimensional (2D), three dimensional (3D), and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with "the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc, where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time) component".

OpenGL Utility Toolkit (GLUT) was developed by Mark Kilgard, it Hides the complexities of differing window system APIs, Default user interface for class projects, Glut routines have prefix glut, Eg- glutCreateWindow(); .

1.2 OpenGL Technology

OpenGL is a graphics application programming interface (API) which was originally developed by Silicon Graphics. OpenGL is not in itself a programming language, like C++, but functions as an API which can be used as a software development tool for graphics applications. The term Open is significant in that OpenGL is operating system independent. GL refers to graphics language. OpenGL also contains a standard library referred to as the OpenGL Utilities (GLU). GLU contains routines for setting up viewing projection matrices and describing complex objects with line and polygon approximations.

OpenGL gives the programmer an interface with the graphics hardware. OpenGL is a low-level, widely supported modeling and rendering software package, available on all platforms. It can be used in a range of graphics applications, such as games, CAD design, modeling.

OpenGL is the core graphics rendering option for many 3D games, such as Quake 3. The providing of only low-level rendering routines is fully intentional because this gives the programmer a great control and flexibility in his applications. These routines can easily be used to build high-level rendering and modeling libraries. The OpenGL Utility Library (GLU) does exactly this, and is included in most OpenGL distributions! OpenGL was originally developed in 1992 by Silicon Graphics, Inc, (SGI) as a multi-purpose, platform independent graphics API. Since 1992 all of the development of OpenGL.

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use 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. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. The OpenGL

Visualization Programming Pipeline:

OpenGL operates on image data as well as geometric primitives. Simplifies Software Development, Speeds Time-to-Market.

Routines simplify the development of graphics software—from rendering a simple geometric point, line, or filled polygon to the creation of the most complex lighted and texture mapped NURBS curved surface. OpenGL gives software developers access to geometric and image primitives, display lists, modeling transformations, lighting and texturing, anti-aliasing, blending, and many other features. Every conforming OpenGL implementation includes the full complement of OpenGL functions. The well-specified OpenGL standard has language bindings for C, C++, FORTRAN, Ada, and Java. All licensed OpenGL implementations come from a single specification and language binding document and are required to pass a set of conformance tests. Applications utilizing OpenGL functions are easily portable across a wide array of platforms for maximized programmer productivity and shorter time-to-market.

All elements of the OpenGL state—even the contents of the texture memory and the frame buffer—can be obtained by an OpenGL application. OpenGL also supports visualization applications with 2D images treated as types of primitives that can be manipulated just like 3D geometric objects. As shown in the OpenGL visualization programming pipeline diagram above.

1.3 PROJECT DESCRIPTION:

This is a gaming project giving you a whole new experience of fishing in computer graphics. The main objective in this game is to fish using the string which can we moved up and down either by the arrow keys (up and down) or by using the left mouse button. When a fish is trapped or actually when the user coincide the area of the fish with the end of the string makes the fish disappear and adds the point to the score all of these have to be done with a minute.

1.4 OpenGl Functions Used:

This project is developed using visual studio 2015 and this project is implemented by making extensive use of library functions offered by graphics package of OpenGl, a summary of those functions follows:

1.4.1 **glBegin() :**

Specifies the primitives that will be created from vertices presented between glBegin and subsequent glEnd.. GL_POLYGON, GL_LINE_LOOP etc.

In the project we have used to create fish ,boat etc.

1.4.2 **glEnd(void) :**

It ends the list of vertices.

1.4.3 **glPushMatrix() :**

*void**glPushMatrix**(void)*

glPushMatrix pushes the current matrix stack down by one level, duplicating the current matrix.

Used in prints function which prints the stroke character in a desired position and even scale the input string.

1.4.4 **glPopMatrix() :**

*void**glPopMatrix(**void)*

glPopMatrixpops the top matrix off the stack, destroying the contents of the popped matrix.Initially, each of the stacks contains one matrix, an identity matrix.

Used in prints function.

1.4.5 **glTranslate*() :**

*void**glTranslate*****(GLdoublex, GLdoubley,GLdouble z)*

Translation is an operation that displaces points by a fixed distance in a given direction. *Parameters x*, *y*, *z* specify the *x*, *y*, and *z* coordinates of a translation vector. Multiplies current matrix by a matrix that translates an object by the given x, y and z-values.

*=>f for float and d for decimal

Used in prints function.

1.4.6 **glClear() :**

*void**glClear**(GLbitfield mask)*

glClear takes a single argument that is the bitwise *or* of several values indicating which buffer is to be cleared. GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, GL_ACCUM_ BUFFER_BIT, and GL_STENCIL_BUFFER_BIT.Clears the specified buffers to their current clearing values.

1.4.7 **glClearColor() :**

*void**glClearColor**(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)*

Sets the current clearing color for use in clearing color buffers in RGBA mode. The red, green, blue, and alpha values are clamped if necessary to the range [0,1]. The default clearing color is (0, 0, 0, 0), which is black.

1.4.8 **glMatrixMode() :**

*void**glMatrixMode**(GLenum mode)*

It accepts three values GL_MODELVIEW, GL_PROJECTION and GL_TEXTURE. It specifies which matrix is the current matrix. Subsequent transformation commands affect the specified matrix.

1.4.9 **glutInitWindowPosition() :**

*void**glutInitWindowPosition**(int x, int y);*

This API will request the windows created to have an initial position. The arguments x, y indicate the location of a corner of the window, relative to the entire display.

1.4.10 **glLoadIdentity() :**

*void**glLoadIdentity**(void);*

It replaces the current matrix with the identity matrix.

1.4.11 **glutInitWindowSize() :**

*void**glutInitWindowSize**(int width, int height);*

The API requests windows created to have an initial size. The arguments width and height indicate the window’s size (in pixels). The initial window size and position are hints and may be overridden by other requests.

1.4.12 **glutInitDisplayMode**

void ***glutInitDisplayMode****(unsigned int mode);*

Specifies the display mode, normally the bitwise OR-ing of GLUT display mode bit *masks.*This API specifies a display mode (such as RGBA or color-index, or single or double-buffered) for windows.

1.4.13 **glFlush() :**

*void****glFlush****(void);*

The glFlush function forces execution of OpenGL functions in finite time.

1.4.14 **glutCreateWindow() :**

*int****glutCreateWindow****(char *name);*

The parameter *name* specifies any name for window and is enclosed in double quotes. This opens a window with the set characteristics like display mode, width, height, and so on.The string name will appear in the title bar of the window system. The value returned is a unique integer identifier for the window. This identifier can be used for controlling and rendering to multiple windows from the same application.

1.4.15 **glutDisplayFunc() :**

*void****glutDisplayFunc****(void (*func)(void))*

Specifies the new display callback function. The API specifies the function that’s called whenever the contents of the window need to be redrawn. All the routines need to be redraw the scene are put in display callback function.

1.4.16 **glVertex2f**

*void****glVertex2f****(GLfloatx,GLfloat y);*

x Specifies the x-coordinate of a vertex.

y Specifies the y-coordinate of a vertex.

The glVertex function commands are used within glBegin/glEnd pairs to specify point, line, and polygon vertices. The current color, normal, and texture coordinates are associated with the vertex when glVertex is called. When only x and y are specified, z defaults to 0.0 and w defaults to 1.0. When x, y, and z are specified, w defaults to 1.0.

1.4.17 **glColor3f**

void glColor3f(GLfloat red, GLfloat green, GLfloat blue);

PARAMETERS:

1. Red: The new red value for the current color.
2. Green: The new green value for the current color.
3. Blue: The new blue value for the current color.

Sets the current color.

1.4.18 **glColor3b();**

void glColor3b(GLbyte red, GLbyte green, GLbyte blue);

1.4.19 **glutInit():**

*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);

glutInit is used to initialize the GLUT library.

1.4.20 **glutMainLoop():**

void*glutMainLoop(void);*

glutMainLoop();

glutMainLoop enters the GLUT event processing loop.

1.4.21 **glutStrokeCharacter();**

Glutstrokecharacter renders a stroke character using openGL.

Usage:

Void glutstrokecharacter(void *font, int character);

Font

Stroke font to use.

Character

Character to render(not confined to 8 bits).

1.4.23 **glutswapbuffer();**

glutSwapBuffers swaps the buffers of the *current window* if double buffered.

Usage

void glutSwapBuffers(void)

Chapter-2

REQUIREMENTS SPECIFICATION

2.1 Hardware requirements:

- ❖ Pentium or higher processor.
- ❖ 128 MB or more RAM.
- ❖ A standard keyboard, and Microsoft compatible mouse
- ❖ VGA monitor resolution 1024*768 and greater.
- ❖ Hard Disk Space: 4.0 GB

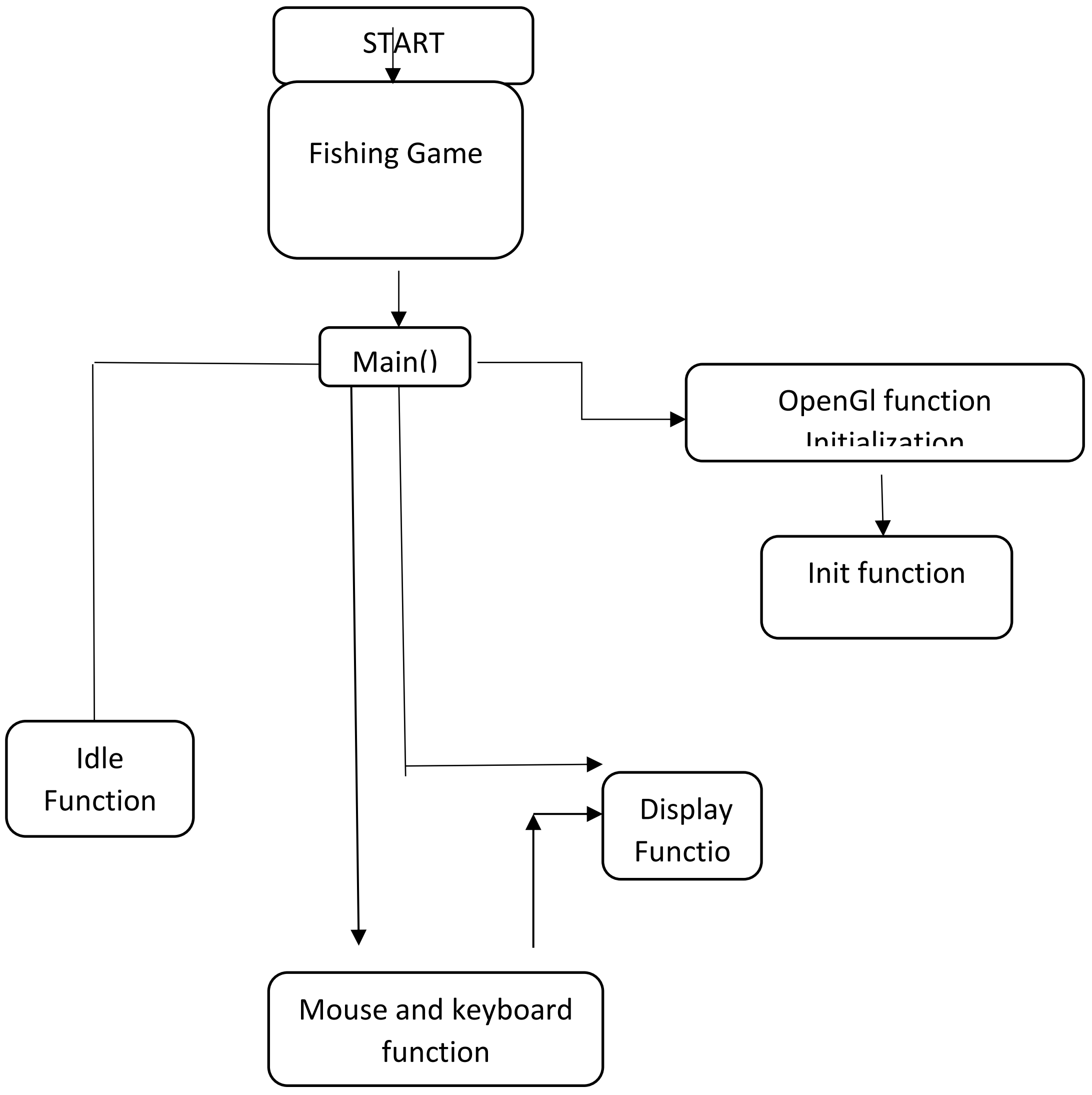
2.2 Software requirements:

- ❖ The graphics package has been designed for OpenGL; hence the machine must have Dev C++.
- ❖ Software installed preferably 6.0 or later versions with mouse driver installed.
- ❖ GLUT libraries, Glut utility toolkit must be available.
- ❖ Operating System**:** Windows
- ❖ Version of Operating System**:** Windows XP, Windows NT and Higher
- ❖ The package is implemented using Microsoft visual C++, under the windows platform. OpenGL and associated toolkits are used for the package development.
- ❖ OpenGL , a software interface for graphics hardware with built in graphics libraries like glut and glut32, and header files like glut.h.

Chapter-3

INTERFACE

3.1 Flow Diagram:



The flow diagram of the Fishing Master

OpenGL is a software tool for developing the graphics objects. OpenGL library called GLUT i.e. Graphics Library Utility toolkit supports graphics system with the necessary modelling and rendering techniques. The Lighting system is a technique for displaying graphic objects on the monitor and displaying the light effects. It provides the following functionalities.

3.2 Initialization

This function is the initial stage of the system where the system initializes the various aspects of the graphics system based on the user requirements, which include Command line processing, window system initialization and also the initial window creation state is controlled by these routines.

3.3 Event Processing

This routine enters GLUT's event processing loop. This routine never returns, and it continuously calls GLUT callback as and when necessary. This can be achieved with the help of the callback registration functions. These routines register callbacks to be called by the GLUT event processing loop.

Chapter-4

IMPLEMENTATION

Step 1: The console window appears and user need to select the option.

Step 2: When the game is started the user can hook the fish by using the keyboard and mouse buttons assigned.

4.1 Source Code

4.1.1 The header files are:

```
#include <windows.h>
#include<stdio.h>
#include<iostream>
#include<math.h>
#include<GL/glut.h>
#include<string.h>
```

4.1.2 The Idle Function:

```
void idle(){
    if(x1 == 0)
        for(int k = 0; k <12 ; k++){
            h[0][k]=rand()%12;
            h[1][k]=rand()%12;
            h[2][k]=rand()%12;
            h[3][k]=rand()%12;
            for(int k1 = 0; k1 <12 ; k1++)
            {
                col[0][k][k1]=(rand()%10)/10.0;
                col[1][k][k1]=(rand()%10)/10.0;
                col[2][k][k1]=(rand()%10)/10.0;
                col[3][k][k1]=(rand()%10)/10.0;
            }
        }
    x1+=.1;
    x2+=.1;
    x3+=.1;
    x4+=.1;
    cnt+=.5;
    counter = cnt;
    rp+=ropeactive;
    chai1=chai;

    if(x1>1024+12*20)
    {
        i1=0;
        for(int u =0; u < 12; u++)
            flag[0][u]=0;
        x1=-341;
        counter=0;
    }
    if(x2>1024+12*20)
    {
        i2=0;
        for(int u =0; u < 12; u++)
            flag[1][u]=0;
        x2=-341;
    }
    if(x3>1024+12*20)
    {
        i3=0;
        for(int u =0; u < 12; u++)
            flag[2][u]=0;
        x3=-341;
    }
```

```
    if(x4>1024+12*20)
    {
        i4=0;
        for(int u =0; u < 12; u++)
            flag[3][u]=0;
        x4=-341;
    }
}
```

4.1.3 The Init Function:

```
void init()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(1.0,0.0,0.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,1024,0,768);


}
```

4.1.4 The code for drawing and rotating the 3D object:

```
void draw()
{
glPushMatrix();
glTranslatef(0,-40.00,-105);
    //glColor3f(1.0,0.23,0.27);
glScalef(0.4,0.4,0.4);
glRotatef(imagerot,0,1,0);
glCallList(image);
glPopMatrix();
imagerot=imagerot+0.2;
if(imagerot>360)imagerot=imagerot-360;
}
```

4.1.5 Boat ,Water ,Rope & Fish functions:

```
void ship(int x,int y)
{
        int m=545, n=700, r=40, i;
            glBegin(GL_POLYGON);
            glColor3f(0.30,0.06,0.42);
            glVertex2f(20,530+7*sin((counter%360)*c));
            glColor3f(0.30,0.06,0.42);
            glVertex2f(20,512+7*sin((counter%360)*c));
        glColor3f(.0,0.0,0.0);
            glVertex2f(82,432+7*sin((counter%360)*c));
        glColor3f(0.0,0.0,0.0);
        glVertex2f(438,432+7*sin((counter%360)*c));
        glColor3f(0.32,0.06,0.42);
        glVertex2f(584,512+7*sin((counter%360)*c));
        glColor3f(0.30,0.06,0.42);
            glVertex2f(600,530+7*sin((counter%360)*c));
            glEnd();

            glBegin(GL_POLYGON);
            glColor3f(0.0,0.0,0.0);
            glVertex2f(60,530+7*sin((counter%360)*c));
            glVertex2f(70,530+7*sin((counter%360)*c));
            glVertex2f(70,730+7*sin((counter%360)*c));
            glVertex2f(60,730+7*sin((counter%360)*c));
        glEnd();

        glBegin(GL_POLYGON);          /*MANNNNNNNNNNNNNN*/
            glColor3f(1.0,0.0,0.0);
            glVertex2f(540,540+7*sin((counter%360)*c));
            glVertex2f(550,540+7*sin((counter%360)*c));
```

```
    glVertex2f(550,670+7*sin((counter%360)*c));
    glVertex2f(540,670+7*sin((counter%360)*c));
glEnd();

  glBegin(GL_POLYGON);          /*HANDS*/
    glColor3f(1.0,0.0,0.0);
    glVertex2f(550,590+7*sin((counter%360)*c));
    glVertex2f(550,600+7*sin((counter%360)*c));
    glVertex2f(580,640+7*sin((counter%360)*c));
    glVertex2f(580,630+7*sin((counter%360)*c));
glEnd();



 glBegin(GL_POLYGON);          /*LEGS*/
    glColor3f(0.0,1.0,0.0);
    glVertex2f(540,530+7*sin((counter%360)*c));
    glVertex2f(540,540+7*sin((counter%360)*c));
    glVertex2f(580,540+7*sin((counter%360)*c));
    glVertex2f(580,530+7*sin((counter%360)*c));
glEnd();

glBegin(GL_POLYGON);                     /*HEAD*/
glColor3f(0.0,0.0,1.0);
for(i=0;i<360;i++)
glVertex2f(m+r*cos(i*c), n+r*sin(i*c)+7*sin((counter%360)*c));
glEnd();

 glBegin(GL_POLYGON);
    glColor3f(0.0,0.0,0.0);
    glVertex2f(550,700+7*sin((counter%360)*c));                  /*...............................EYES*/
    glVertex2f(553,700+7*sin((counter%360)*c));
    glVertex2f(553,705+7*sin((counter%360)*c));
```

```
        glVertex2f(550,705+7*sin((counter%360)*c));
        glEnd();
         glBegin(GL_POLYGON);
        glColor3f(0.0,0.0,0.0);
        glVertex2f(570,700+7*sin((counter%360)*c));
        glVertex2f(573,700+7*sin((counter%360)*c));
        glVertex2f(573,705+7*sin((counter%360)*c));
        glVertex2f(570,705+7*sin((counter%360)*c));
        glEnd();

        glBegin(GL_LINES);                    /*MOUTH*/
        glColor3f(0.0,0.0,0.0);
        glVertex2d(550,680+7*sin((counter%360)*c));
        glVertex2d(565,680+7*sin((counter%360)*c));
        glEnd();

      glBegin(GL_POLYGON);          /*FISHING POLE*/
        glColor3f(0.0,0.0,0.0);
        glVertex2f(580,640+7*sin((counter%360)*c));
        glVertex2f(580,630+7*sin((counter%360)*c));
        glVertex2f(620,660+7*sin((counter%360)*c));
        glVertex2f(620,670+7*sin((counter%360)*c));
      glEnd();
      glBegin(GL_POLYGON);
        glColor3f(0.0,0.0,0.0);
        glVertex2f(620,660+7*sin((counter%360)*c));
        glVertex2f(620,670+7*sin((counter%360)*c));
        glVertex2f(750,750+7*sin((counter%360)*c));
        glEnd();

}
void water1()
```

```
{
   glBegin(GL_POLYGON);
   glColor3f(0.0,0.0,1.0);
   glVertex2f(0,0);
   glVertex2f(0,450);
   for(int i = 0; i <=1024 ; i++)
      glVertex2f(i,450+5*sin(((5*(i-counter/2)%360)*c)));
   glVertex2f(1024,450);
   glVertex2f(1024,0);
   glEnd();
}

void fish(int d,int y1,GLfloat clr[3],int f1,int f2)
{
    GLfloat xs=0.0;
   GLfloat ys = 20.0+y1+14*sin((counter%360)*c) ;
   if((GLfloat)(20.0+d)>= (GLfloat)750.0 && (GLfloat)(d-30.0) <= (GLfloat)750.0 )
   {
      if((GLfloat)ys < (GLfloat)650+rp && (GLfloat)ys > (GLfloat)610+rp)
            {
               flag[f1][f2]=1;
               points+=10;
            }
   }
   glColor3fv(clr);
   glBegin(GL_POLYGON);
   glVertex2f(20+d+xs,15+y1+14*sin((counter%360)*c));
   glVertex2f(25+d+xs,10+y1+14*sin((counter%360)*c));
   glVertex2f(30+d+xs,20+y1+14*sin((counter%360)*c));
   glVertex2f(25+d+xs,35+y1+14*sin((counter%360)*c));

   glVertex2f(20+d+xs,30+y1+14*sin((counter%360)*c));
```

```
    glVertex2f(15+d+xs,28+y1+14*sin((counter%360)*c));
    glVertex2f(10+d+xs,35+y1+14*sin((counter%360)*c));
    glVertex2f(10+d+xs,10+y1+14*sin((counter%360)*c));
    glVertex2f(15+d+xs,17+y1+14*sin((counter%360)*c));
    glEnd();

    glColor3f(0,0,0);
    glBegin(GL_LINE_STRIP);
    glVertex2f(26+d+xs,15+y1+14*sin((counter%360)*c));
    glVertex2f(28+d+xs,20+y1+14*sin((counter%360)*c));
    glVertex2f(26+d+xs,30+y1+14*sin((counter%360)*c));
    glEnd();
}

void rope()
{
    glBegin(GL_LINES);          /*FISHING STRING */
        glVertex2d(750,750+7*sin((counter%360)*c));//change value to top if rod point
        if( rp + 500 > 500)
            rp=0;//change x value to same as top and y to top of water
if(rp+600<0)
        rp=-600;
        glVertex2d(750,650+rp);//change value to same as above +rp
        glEnd();

            glBegin(GL_LINE_STRIP);
            glColor3f(0.0,0.0,0.0);
            glVertex2f(750,650+rp);
            glVertex2f(755,640+rp);
            glVertex2f(760,630+rp);
            glVertex2f(755,620+rp);
            glVertex2f(750,610+rp);
        glEnd();

}
```

4.1.6 The Timer Function :

```
                                void timer(int time)
                                      {
  if(chai1==1)
  {

  if(mins == 0 && sec == 0)
  {
    tf=1;
    glutMouseFunc(NULL);
    glutSpecialFunc(NULL);
  }
  else if(sec == 0)
  {
    mins--;
    sec=59;
  }
  else
    sec--;

}
glutTimerFunc(1000,timer,0);
}
```

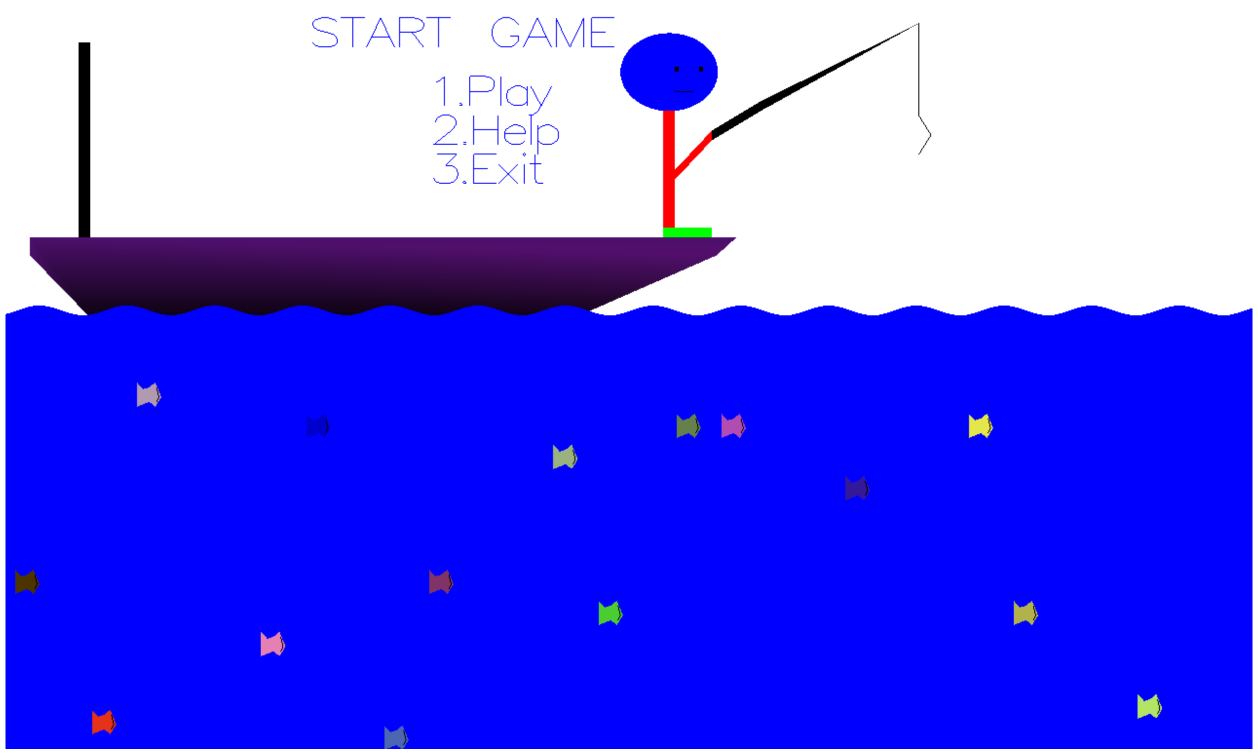
4.1.7 The Keyboard ,Mouse and special key functions:

```
void MS(int btn, int state, int x, int y)
{
   if(btn==GLUT_LEFT_BUTTON&&state==GLUT_DOWN)
      if(ropeactive ==0)
         ropeactive=-5;
      else
         ropeactive*=-1;
}

void KB(unsigned char btn, int x ,int y)
{
   if(btn == 51)
      _exit(0);
      if (btn == 49)
      chai=1;
       if (btn == 50)
      chai=2;
      if(btn==32)
         chai=0;



}

void KB1(int btn, int x ,int y)
{
```

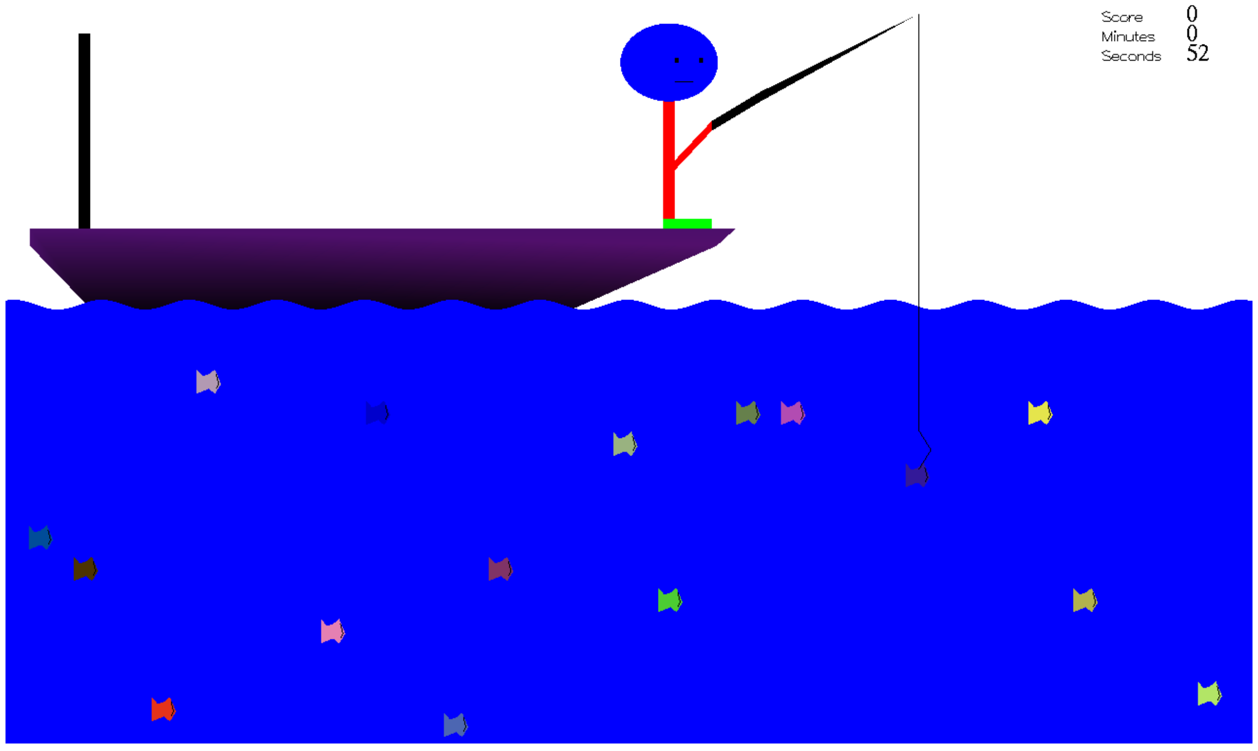
```
int chai=0;

if(btn == GLUT_KEY_DOWN)

    ropeactive = -.5;

if(btn == GLUT_KEY_UP)

    ropeactive = .5;
```

Chapter - 5

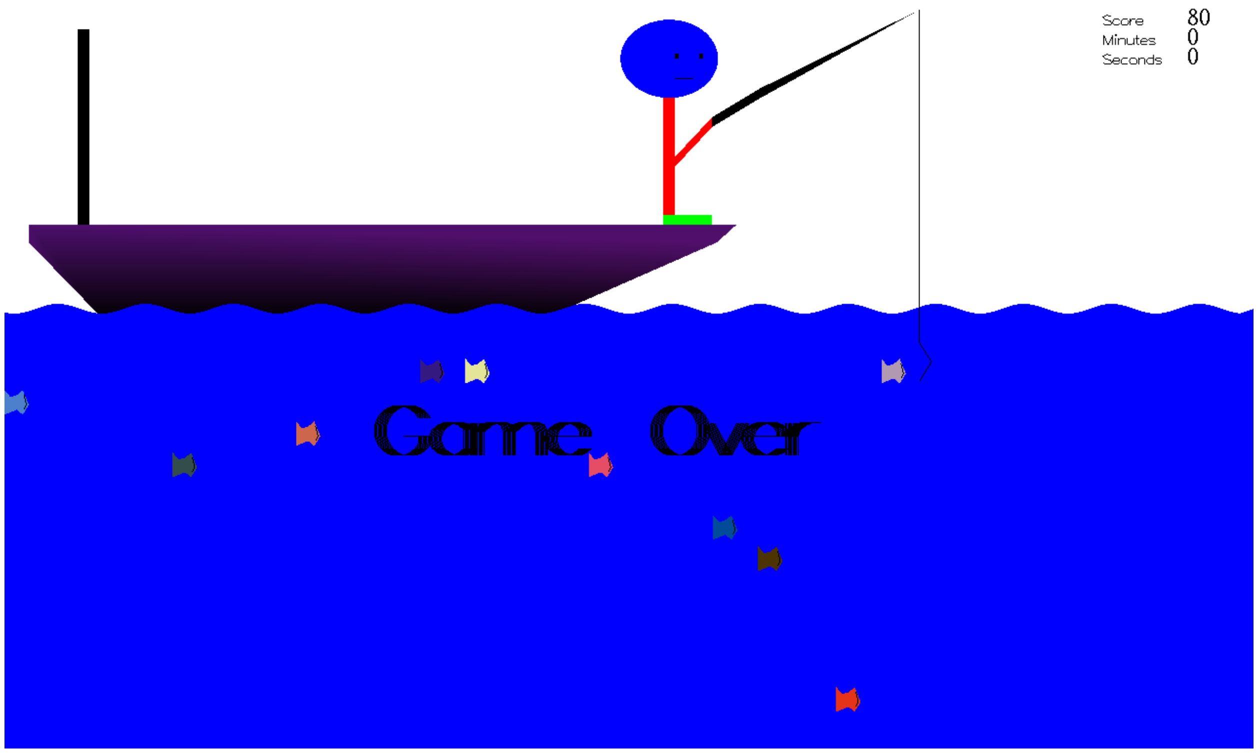
SNAPSHOTS



Main screen with menu .



Gameplay.



GAME OVER .

Chapter 6

CONCLUSION AND FUTURE ENHANCEMENTS

The aim of the program led to the development of adventurous and strategic fishing game play . The game has a time limit which is one minute within which the player have to maximize score by fishing. After one minute the final score of the player will be displayed . The player can catch the fish either by using the UP and DOWN arrows or using the left mouse button. If we press the UP or DOWN arrows key the string keeps on changing the length which allows the Fisherman to catch the fish.

- We can further develop the game in 3d .
- We can try making first person gameplay to make it realistic.

REFERENCES

Books:

[1] The Red Book –OpenGL Programming Guide,6th edition.

[2] Rost , Randi J. : OpenGL Shading Language, Addison-Wesley

[3] Interactive Computer Graphics-A Top Down Approach Using OpenGL, Edward Angel, Pearson-5th edition.