VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELGAVI-590014

Computer Graphics Laboratory
A Mini-Project Report
On

"BLACK HOLE SIMULATION"

Submitted in partial fulfillment of the requirements for the 6th semester of ***Bachelor of Engineering in Computer Science and Engineering*** *of Visvesvaraya Technological University, Belagavi*

by:

MOHIT K	**1RN15CS061**
M BHARADWAJ	**1RN15CS055**

Under the Guidance of:

Mrs. S Mamatha Jajur
Assistant Professor
Dept. of CSE

Mrs. Sampada K S
Assistant Professor
Dept. of CSE

Department of Computer Science and Engineering
RNS Institute of Technology
Channasandra, Dr.Vishnuvardhan Road, Bengaluru-560 098

2017-2018

RNS Institute of Technology
Channasandra, Dr.Vishnuvardhan Road,
Bengaluru-560 098

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

Certified that the mini project work entitled **“Black Hole Simulation”** has been successfully carried out by **Mohit K** bearing USN **1RN15CS061** and **M Bharadwaj** bearing USN **1RN15CS055**, bonafide students of **RNS Institute of Technology** in partial fulfillment of the requirements for the **6th semester** of **Bachelor of Engineering** in **Computer Science and Engineering** of **Visvesvaraya Technological University**, Belagavi, during the academic year 2017-18. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the Computer Graphics laboratory requirements of 6th semester BE, CSE.

Mrs. S Mamatha Jajur	**Mrs. Sampada K S**	**Dr. G T Raju**
Assistant Professor	**Assistant Professor**	**Prof. and Head**
Dept. of CSE	**Dept. of CSE**	**Dept. of CSE**

External Viva:

Name of the Examiners	**Signature with Date:**
1.	
2.	

ACKNOWLEDGEMENTS

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all.

We would like to thank **Dr. H N Shivashankar**, Director, RNSIT, Bangalore, for his moral support towards completing our project.

We are grateful to **Dr. M K Venkatesha,** Principal, RNSIT, Bangalore, for his support towards completing our project.

We would like to thank **Dr. G T Raju**, Dean of Engg., Prof & Head, Department of Computer Science & Engineering, RNSIT, Bangalore, for his valuable suggestions and expert advice.

We deeply express our sincere gratitude to our guides **Mrs. S Mamatha Jajur, Assistant Prof, Department of CSE, RNSIT, Bangalore** and **Mrs. Sampada K S**, **Assistant Prof, Department of CSE, RNSIT, Bangalore,** for their able guidance, regular source of encouragement and assistance throughout this project.

We would like to thank all the teaching and non-teaching staff of Department of Computer Science & Engineering RNSIT, Bangalore for their constant support and encouragement.

Date:	**Mohit K 1RN15CS061**
Place: Bengaluru	**M Bharadwaj 1RN15CS055**

ABSTRACT

This project is about the simulation of Black Hole .We are implementing it using different primitives available in Open GL library and combining them in a required manner.

It highlights the key features of Open GL transformation functions and its usage in a project. This project consists of a Solar System with planets revolving around a star. The satellites of some of these planets have also been displayed. The solar system is consumed by a Black Hole which pulls the planets along with their satellites as well as other asteroids in space into it. It illustrates the role of different call back functions that provide easy way to accomplish our project in an effective manner.

The project has been implemented using various Open GL transformation functions such as translate , rotate and scale .We have also made use of various other features that are made available by Open GL software package and have been used effectively.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF COMPUTER GRAPHICS

The term computer graphics has been used in a broad sense to describe almost everything on computers that is not text or sound. Typically, the term *computer graphics* refers to several different things:

- The representation and manipulation of image data by a computer
- The various technologies used to create and manipulate images
- The images so produced, and
- The sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.

Today, computers and computer-generated images touch many aspects of daily life. Computer images are found on television, in newspapers, for example in weather reports, in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media such graphs are used to illustrate papers, reports, thesis, and other presentation material.Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D, 3D, 4D, 7D, and animated graphics.

As technology has improved, 3D computer graphics have become more common. 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 component.

1.1.1 HISTORY

In 1959, the TX-2 computer was developed at MIT's Lincoln Laboratory. The TX-2 integrated a number of new man-machine interfaces. A light pen could be used to draw sketches on the computer using Ivan Sutherland's revolutionary Sketchpad software.[4] Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, save them and even recall them later. The light pen itself had a small photoelectric cell in its tip. This cell emitted an electronic pulse whenever it was placed in front of a computer screen and the screen's electron gun fired directly at it. By simply timing the electronic pulse with the current location of the electron gun, it was easy to pinpoint exactly where the pen was on the screen at any given moment. Once that was determined, the computer could then draw a cursor at that location. Also in 1961 another student at MIT, Steve Russell, created the first video game, E. E. Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-giro gravity attitude control system" in 1963.

During 1970s, the first major advance in 3D computer graphics was created at UU by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are "behind" the object from the viewer's perspective, and thus should be "hidden" when the computer creates (or renders) the image.

In the 1980s, artists and graphic designers began to see the personal computer, particularly the Commodore Amiga and Macintosh, as a serious design tool, one that could save time and draw more accurately than other methods. In the late 1980s, SGI computers were used to create some of the first fully computer-generated short films at Pixar. The Macintosh remains a highly popular tool for computer graphics among graphic design studios and businesses. Modern computers, dating from the 1980s often use graphical user interfaces (GUI) to present data and information with symbols, icons and pictures, rather than text. Graphics are one of the five key elements of multimedia technology.

3D graphics became more popular in the 1990s in gaming, multimedia and animation. In 1996, Quake, one of the first fully 3D games, was released. In 1995, Toy Story, the first full-length computer-generated animation film, was released in cinemas worldwide. Since then, computer graphics have only become more detailed and realistic, due to more powerful graphics hardware and 3D modeling software.

1.1.2 APPLICATIONS

The applications of computer graphics can be divided into four major areas:

- **Display of information:** Computer graphics has enabled architects, researchers and designers to pictorially interpret the vast quantity of data. Cartographers have developed maps to display the celestial and geographical information. Medical imaging technologies like Computerized Tomography (CT), Magnetic Resonance Imaging (MRI), Ultrasound, Positron Emission Tomography (PET) and many others make use of computer graphics.
- **Design:** Professions such as engineering and architecture are concerned with design. They start with a set of specification; seek cost-effective solutions that satisfy the specification. Designing is an iterative process. Designer generates a possible design, tests it and then uses the results as the basis for exploring other solutions. The use of interactive graphical tools in Computer Aided Design (CAD) pervades the fields including architecture, mechanical engineering, and the design of very-large-scale integrated (VLSI) circuits and creation of characters for animation.
- **Simulation and animation:** Once the graphics system evolved to be capable of generating sophisticated images in real time, engineers and researchers began to use them as simulators. Graphical flight simulators have proved to increase the safety and to reduce the training expenses. The field of virtual reality (VR) has opened many new horizons. A human viewer can be equipped with a display headset that allow him/her to see the images with left eye and right eye which gives the effect of stereoscopic vision. This has further led to motion pictures and interactive video games.
- **User interfaces:** Computer graphics has led to the creation of graphical user interfaces (GUI) using which even naive users are able to interact with a computer. Interaction with the computer has been dominated by a visual paradigm that includes windows, icons, menus and a pointing device such as mouse. Millions of people are internet users; they access the internet through the graphical network browsers such as Microsoft internet explorer and Mozilla Firefox.

1.2 DESCRIPTION OF PROJECT

This project is a simulation of Black Hole. The project starts with a solar system with planets and satellites .The planets and satellites are representd by wired spheres .The planets revolve around the star . We can adjust the speed of revolution of the planets .It can be viewed in top and side view.By right click we can access a drop down menu which allows us to initiate a Black Hole. On initialization the space matter is then consumed by a Black Hole which pulls everything towards its centre .

1.3 OVERVIEW TO OpenGL

OpenGL (Open Graphics Library) 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. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation.

OpenGL provides a set of commands to render a three dimensional scene. That means you provide the data in an OpenGL-useable form and OpenGL will show this data on the screen (render it). It is developed by many companies and it is free to use. You can develop OpenGL-applications without licensing. OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation. Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

1.4 OpenGL LIBRARIES

Computer Graphics are created using OpenGL, which became a widely accepted standard software system for developing graphics applications. 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.

OpenGL stands for ‘open graphics library’ graphics library is a collection of API’s (Applications Programming Interface). Graphics library functions are:

1. **GL library** (OpenGL in windows) – Main functions for windows.
2. **GLU** (OpenGL utility library) - Creating and viewing objects.
3. **GLUT** (OpenGL utility toolkit)- Functions that help in creating interface of windows

OpenGL draws *primitives*—points, line segments, or polygons—subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. These libraries are included in the application program using preprocessor directives

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

OpenGL User Interface Library (**GLUI**) is a C++ user interface library based on the OpenGL Utility Toolkit (GLUT) which provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications. It is window and operating system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management.

The **OpenGL Utility Library** (**GLU**) is a computer graphics library. It consists of a number of functions that use the base OpenGL library to provide higher-level drawing routines from the more primitive routines that OpenGL provides. It is usually distributed with the base OpenGL package.

CHAPTER 2

SYSTEM DESIGN

2.1 HARDWARE REQUIREMENTS

The Hardware Requirements are very minimal and the program can be run on most of the machines.

Processor: Intel Core i3 processor

Processor Speed: 1.70 GHz

RAM: 4 GB

Storage Space: 40 GB

Monitor Resolution: 1024x768 or 1336x768 or 1280x1024

2.2 SOFTWARE REQUIREMENTS

Operating System: Windows

IDE: Microsoft Visual Studio with C++ (version 6)

OpenGL libraries, Header Files which includes GL/glut.h, Object File Libraries, glu32.lib,opengl32.lib,glut32.lib,DLLfiles,glu32.dll,glut32.dll,opengl32.dll.

2.3 Functional Requirements

First we need to create the star at the centre . We aceive this through glutSolidSphere function .We use the translate function to position it.Then using push and pop matrix as well as glTranslate and glRotate function we position and give the rotation and revelution of planets and satellites. They are represented using glutWiredSphere. We then use for loop as well as translate feature of OpenGL to simulate the black hole.

CHAPTER 3

SYSTEM DESIGN

3.1 MODULES

The description of all the functions used in the program is given below:

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

This function requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE).

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

This specifies the initial position of top-left corner of the windows in pixels.

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

  This function specifies the initial height and width of the window in pixels.

- **glTranslatef(tx, ty, tz);**

  Translation refers to moving an object to a different position on screen.

  Formula: X = x + tx

  Y = y + ty

  where tx and ty are translation coordinates

- **glRotatef (A, x, y, z);**

  Rotation refers to rotating a point.

  Formula: X = xcosA - ysinA

  Y = xsinA + ycosA,

  A is the angle of rotation.

  The above formula will rotate the point around the origin.

  To rotate around a different point, the formula:

X = cx + (x-cx)*cosA - (y-cy)*sinA,

Y = cx + (x-cx)*sinA + (y-cy)*cosA,

cx, cy is centre coordinates,

A is the angle of rotation.

- **glScalef(float x, float y, float z);**

  Scaling refers to zooming in and out an object in different scales across axes.

  Formula: X = x*sx

  Y = y*sy,   sx, sy being scaling factors.

- **glLight*(light,pname,param);**

  light

  Specifies a light. The number of lights depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form GL_LIGHT i, where i ranges from 0 to the value of GL_MAX_LIGHTS - 1.

  pname

  Specifies a single-valued light source parameter for light. GL_SPOT_EXPONENT, GL_SPOT_CUTOFF, GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, and GL_QUADRATIC_ATTENUATION are accepted.

  param

  Specifies the value that parameter pname of light source light will be set to.

- **void glutCreateWindow (char *title)**

This function creates a window on the display the string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

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

This function registers the display func that is executed when the window needs to be redrawn.3

- **void glClearColor(GLclampfr,GLclampf g, GLclampfb,GLclampf a)**

This sets the present RGBA clear colour used when clearing the colour buffer. Variables of type GLclampf are floating point numbers between 0.0 and 1.0.

- **void glClear(GLbitfield mask)**

It clear buffers to present values. The value of mask is determined by the bitwise OR of options GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT

- **void glutPostRedisplay ()**

This function requests that the display callback be executed after the current callback returns.

- **void glutReshapeFunc (void *f (int width, int height)**

This function registers the reshape callback function f. The callback function returns the height and width of the new window. The reshape callback invokes a display callback.

- **void glViewport (int x, int y, GLsizei width, GLsizei height)**

This function specifies a width*height viewport in pixels whose lower left corner is at
(x, y) measured from the origin of the window.

- **void glMatrixMode (GLenum mode)**

This function specifies which matrix will be affected by subsequent transformations. Mode can be GL_MODEL_VIEW, GL_PROJECTION, GL_TEXTURE.

- **void glLoadIdentity ()**

This function sets the current transformation matrix to an identity matrix.

- **void gluOrtho2D(GLdoubleleft, GLdouble right, GLdouble bottom, GLdouble top)**

This function defines a two-dimensional viewing rectangle in the plane z=0.

- **void glutMouseFunc (void *f (int button, int state, int x, int y)**

This function registers the mouse callback function f. The callback function returns the button(GLUT_LEFT_BUTTON,GLUT_MIDDLE_BUTTON,GLUT_RIGHT_BUTTON), the state of the button after the event (GLUT_UP, GLUT_DOWN), and the position of the mouse relative to the top-left corner of the window.

- **void glVertex3f(TYPE xcoordinate, TYPE ycoordinate, TYPE zcoordinate)**

void glVertex3fv(TYPE *coordinates)
This specifies the position of a vertex in 3 dimensions. If v is present, the argument is a pointer to an array containing the coordinates.

- **void glBegin(glEnum mode)**

This function initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL_POINTS, GL_LINES and GL_POLYGON.

- **void glEnd()**

This function terminates a list of vertices.

- **void glutMainLoop()**

This function causes the program to enter an event processing loop. It should be the last statement in main.

- **void glPushMatrix(void);**
  Pushes all matrices in the current stack down one level. The current stack is determined by glMatrixMode(). The topmost matrix is copied, so its contents are duplicated in both the top and second-from-the-top matrix. If too many matrices are pushed, an error is generated.
  Void glRotate{fd} (TYPE angle, TYPE x, TYPE y, TYPE z);
  voidglTranslate{fd} (TYPE x, TYPE y, TYPE z);
  Void glScale {fd} (TYPE x, TYPE y, TYPE z);

- **void glutWireSphere(GLdouble radius,GLint slices,GLint stacks);**

radius
The radius of the sphere.
slices
The number of subdivisions around the Z axis (similar to lines of longitude).
stacks
The number of subdivisions along the Z axis (similar to lines of latitude).
Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

- **void glutSolidSphere(GLdouble radius,GLint slices,GLint stacks);**
  radius
  The radius of the sphere.
  slices
  The number of subdivisions around the Z axis (similar to lines of longitude).
  stacks
  The number of subdivisions along the Z axis (similar to lines of latitude).
  Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

- **glLoadName(ID) :**
  To replace the top of the integer-ID name stack for picking operations

- **Void glPopMatrix (void);**
  Pops the top matrix off the stack, destroying the contents of the popped matrix. What was the second-from-the-top matrix becomes the top matrix. The current stack is determined byglMatrixMode(). If the stack contains a single matrix, calling glPopMatrix() generates an error.

- **glSelectBuffer(size,pickbuff)** : To set up a pick-buffer array

- **glRenderMode(GL_SELECT) :** To activate OpenGL picking operations

- **glInitNames() :** To activate the integer-ID name stack for the picking operations

- **glPushName(ID) :** To place an unsigned interger value on the stack

- **glGet** :** Query function to copy specified state values into an array(requires specification of data type,symbolic name of a state parameter and an array pointer)

- **gluPickMatrix(xPick,yPick,widthPick,heightPick,vpArray) :** To define a pick window within a selected viewport

- **glPopMatrix()** : To destroy the matix on top of the stack and to make the second matrix on the stack become the current matrix

- **glutMouseFunc(mousefcn)** : Specify a mouse callback function that is to be invoked when a mouse button is pressed

- **glutMotionFunc(motionfcn) :** Specify a mouse callback function that is to be invoked when the mouse cursor is moved while a button is pressed

- **glutKeyboardFunc(keyboardfcn)** : Specify a keyboard callback function that is to be invoked when a standard key is pressed

- **glutCreateMenu(menuFunc) :** To create a popup menu and specify the procedure to be invoked when a menu item is selected

- **glutAddMenuEntry(charString,menuItemNumber) :** To list the name and position for each option

- **glutAttachMenu(button)** : To specify a mouse button that is to be used to select a menu option

CHAPTER 4

IMPLEMENTATION

We will place a star at the origin of the openGL coordinate system. We will use a predefined wire-frame sphere included in the GLUT. Place the following GLUT and openGL commands between the glPushMatrix( ) and glPopMatrix( ) in the function Display().

```
glColor3ub(120,120,30);           //                                     yellow
glutSolidSphere(0.3,20,20);        //                                       star
```

The glColor3ub( ) commands sets the drawing color to yellow. The glutWireSphere( ) command defines a sphere with a radius of 0.3, 20 sections in circumference and 20 layers. The number of sections and layers are similar to longitude and latitude lines.

For our first transformation, let's rotate the entire scene so that the axis of the sphere (orientation of the poles) is vertical. Currently the x-axis is extends to the left and right, the y-axis is up and down and the z-axis is into and out of the image (i.e. away from and toward our viewing location). Add the following command in the Display( ) function just after glPushMatrix( ) and before drawing the wire sphere.

```
glRotatef(90.0,1.0,0.0,0.0);      // flip scene 90 degrees about the x-axis
```

Now we are ready to include some animation to our scene. We would like to change the speed of rotation of this wire sphere by pressing the left and right arrows keys. Since we want to set the speed of rotation using the arrow keys the sphere should continue to rotate at the specified speed after we have pressed a key. Pressing the right arrow key should increase the rotation speed and then pressing the left arrow key should reduce the speed of rotation.

There are several modifications needed to implement this animation. First we will define variables for the angle of rotation, the speed of revolution (which is the rate of change in the angle of rotation), and the amount of change in speed we want for each key press.

```
float sunrev   =   0.0;     //   the   amount   of   revolution   of   the   sun
float delsunrev = 0.0; // the amount of change in the revolution (rotation) per
frame                                   of                                animation
float delang = 0.01;   // the amount of change in the rate of rotation per key
press
```

These variables should be declared globally in the same region of the program as the integer fullscreen. The function Display( ) should now appear similar to the code shown here.

```
void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT            |           GL_DEPTH_BUFFER_BIT);
    glPushMatrix();                    // push drawing and transforms onto
stack
    glRotatef   (90.0,1.0,0.0,0.0);     //   flips   scene   90   degrees
    glRotatef(sunrev,0.0,0.0,1.0);     //   rotates   "sun"   about   z-axis
    glColor3ub(120,120,30);            //    sets    color    to    yellow
    glutWireSphere(0.3,20,20);         //       draw        the        "sun"
    glPopMatrix();                     //   pop   matrix   to   draw   scene
    glutSwapBuffers();                 // swap display frames to show new
scene
}
```

In the SpecialKeys( ) function we need to include statements to increment or decrement the the value of delsunrev by an amount delang. This value will be used to animate the "sun's" revolution. Modify the GLUT_KEY_LEFT and GLUT_KEY_RIGHT if constructs in the following manner.

```
    if (key == GLUT_KEY_LEFT)
    {
        delsunrev += delang;
    }

    if (key == GLUT_KEY_RIGHT)
    {
        delsunrev -= delang;
    }
```

Finally we need to add the following statement to the animate( ) function.

```
    sunrev += delsunrev;
```

It is imortant to understand that this function has been specified as the function to call when nothing else is happening (i.e. on idle). The glutPostRedisplay( ) command in the animate( ) function ensures that the scene will be rendered each time animate is called. But since animate( ) is called continuously, our scene will be updated and redrawn so long as the program is running.

We have one more issue to resolve before we test our animation. Since the value of the rotation angle sunrev is being incremented continuously by the animate( ) function, we need to make sure that its magnitude is kept between 0.0 and 360.0 degrees. To do this we will use CheckRot( ) to subtract or add 360.0 to the current value of sunrev as needed.. Modify CheckRot( ) to as shown below to perform this operation.

```
void CheckRot()
{
    if(sunrev>360.0)
        sunrev-=360.0;
    if(sunrev<0.0)
        sunrev+=360.0;
}
```

At this point the animate( ) function should look similar to the following:

```
void animate(void)
{
    sunrev+=delsunrev;
    CheckRot();
    glutPostRedisplay();
}
```

At first, it may seem that we have gone to a lot of trouble to spin a wire sphere. However, it is important to keep in mind that this animation is able to run smoothly while we interact with it. Anyone who has attempted to build a multi-threaded interactive GUI application will appreciated the power of the GLUT callback function interface. When you have successfully animated the spinning of the "sun" sphere, continue to the next step.

Adding a planet - As discussed , the transformation matrices are pushed onto a stack which means that they are processed in reverse order. We need to make the planet spin on its own axis (revolve) and we need to make it orbit the sun. To model an orbit we must translate the planet by an amount equal to the orbital radius and then rotate the displaced planet around the origin.

Modify the Display( ) function to include a second wire sphere as shown below.

```
    glRotatef(90.0,1.0,0.0,0.0);

    glRotatef(sunrev,0.0,0.0,1.0);     // revolves the sun on its axis
    glColor3ub(120,120,30);            // set color to yellow
    glutWireSphere(0.3,20,20);         // draw the sun

    glRotatef(planetorb,0.0,0.0,1.0); // orbits the planet around the
sun
    glTranslatef(1.5,0.0,0.0);         // sets the radius of the orbit
    glRotatef(planetrev,0.0,0.0,1.0); // revolves the planet on its axis
    glColor3ub(30, 120, 30);          // set color to green
    glutWireSphere(0.1,20,20);        // draw the planet
```

The variables for planetorb and planetrev should be created and handled in the same manner as sunrev. Be sure to include the appropriate code in the CheckRot( ), SpecialKeys( ), and animate( ) functions. Once you have successfully implemented your planet.

Now we want to include a smaller sphere that revolves on its axes and obits the planet while the planet orbits the sun. In order to provide a bit more interest to the model we can give each revolution and orbital rates different magnitudes. As an example, compare the rates provided in the SpecialKeys( ) function listed below.

```
void SpecialKeys(int key, int x, int y)
{
    if (key==GLUT_KEY_LEFT)
    {
        delplanetrev+=30.0*delang;
        delmoonorb+=5.0*delang;
        delmoonrev+=50.0*delang;
        delplanetorb+=1.0*delang;
        delsunrev+=delang;
    }

    if (key==GLUT_KEY_RIGHT)
    {
        delplanetrev-=30.0*delang;
        delmoonorb-=5.0*delang;
        delmoonrev-=50.0*delang;
        delplanetorb-=1.0*delang;
        delsunrev-=delang;
    }
}
```

Tilting the Solar System - As a final touch you may want to tilt the entire solar system by pressing the up arrow and down arrow keys. In this case we don't want the solar system to continue to change its pitch after we release the arrow keys so we will use the SpecialKeys( ) function to change the tilt angle directly as shown below.

```
    if (key==GLUT_KEY_UP)
        tilt+=1.0;

    if (key==GLUT_KEY_DOWN)
        tilt -= 1.0;
```

The tilt of the solar system should directly follow the 90 rotation we placed at the top of the Display( ) function.

The following code snippet will create the black hole :

```
        //glPushMatrix();
                glLoadIdentity();

                //glPushMatrix();
                glTranslatef(0.0, 0.0, -8.0);
                glRotatef(90.0*tilt, 1.0, 0.0, 0.0);
                glRotatef(sunrev, 0.0, 0.0, 1.0);

                //glPushMatrix();
                //black hole
                glColor3f(0, 0, 0);
                if (r < 100){
                        for (unsigned int k = 0; k < 5000000; k++);
                        r += 0.002;
                }
                glutSolidSphere(r, 20, 20);
                if (f>0)
                        f -= 0.01;
                if (f1 > 0)
                        f1 -= 0.001;
                if (f2 > 0)
                        f2 -= 0.01;
                if (f3 > 0)
                        f3 -= 0.01;
                if (f4 > 0)
                        f4 -= 0.01;
                if (f5 > 0)
                        f5 -= 0.01;
                if (f6 > 0)
                        f6 -= 0.01;
                if (f7 > 0)
                        f7 -= 0.01;
                if (f8 > 0)
                        f8 -= 0.01;
                if (r == 10)
                        choice = 2;
                //asteroid
                glPushMatrix();
                glRotatef(venuorb, 0.0, 0.0, 1.0);
                glTranslatef(f6, f5, f5);
                glRotatef(venurev, 0.0, 0.0, 1.0);
                glColor3f(1, 1, 0);
                glScalef(0.1, 0.1, 0.1);
                glutSolidIcosahedron();
                glPopMatrix();
                glPushMatrix();
                glRotatef(venuorb, 0.0, 0.0, 1.0);
                glTranslatef(f7, f8, f8);
                glRotatef(venurev, 0.0, 0.0, 1.0);
                glColor3f(1, 1, 0);
                glScalef(0.2, 0.2, 0.2);
                glutSolidIcosahedron();
                glPopMatrix();
                glPushMatrix();
                glRotatef(venuorb, 0.0, 0.0, 1.0);
                glTranslatef(f5,0, f4);
                glRotatef(venurev, 0.0, 0.0, 1.0);
                glColor3f(1, 1, 0);
                glScalef(0.1, 0.1, 0.1);
                glutSolidIcosahedron();
                glPopMatrix();
                glPushMatrix();
                glRotatef(venuorb, 0.0, 0.0, 1.0);
```

```
glTranslatef(f3, f4,0);
glRotatef(venurev, 0.0, 0.0, 1.0);
glColor3f(0, 0.1, 0.9);
glScalef(0.05, 0.05, 0.05);
glutSolidIcosahedron();
glPopMatrix();

glPushMatrix();
glRotatef(venuorb, 0.0, 0.0, 1.0);
glTranslatef(f, f1, 0.0);
glRotatef(venurev, 0.0, 0.0, 1.0);
glColor3f(1, 1, 0);
glutWireSphere(0.1, 10, 10);
glPopMatrix();

glPushMatrix();
glRotatef(merorb, 0.0, 0.0, 1.0);
glTranslatef(f2, 0.0, 0.0);
glRotatef(merrev, 0.0, 0.0, 1.0);
glColor3f(0.5, 0.5, 0);
glutWireSphere(0.2, 10, 10);
glPopMatrix();



glPushMatrix();
glRotatef(planetorb, 0.0, 0.0, 1.0);
glTranslatef(f3, 0.0, 0.0);
glRotatef(planetrev, 0.0, 0.0, 1.0);
glColor3f(0, 1, 0);
glutWireSphere(0.3, 20, 20);

glRotatef(moonorb, 0.0, 0.0, 1.0);
glTranslatef(0.5, 0.0, 0.0);
```

```
glRotatef(moonrev, 0.0, 0.0, 1.0);
glColor3f(.9, .8, .7);
glutWireSphere(0.08, 10, 10);
glPopMatrix();

glPushMatrix();
glRotatef(marorb, 0.0, 0.0, 1.0);
glTranslatef(f4, 0.2, 0.0);
glRotatef(marrev, 0.0, 0.0, 1.0);
glColor3f(1, 0, 0);
glutWireSphere(0.1, 10, 10);
glPopMatrix();

glPushMatrix();
glRotatef(juporb, 0.0, 0.0, 1.0);
glTranslatef(f5, 0.0, 0.0);
glRotatef(juprev, 0.0, 0.0, 1.0);
glColor3f(0.6, 0.1, 0.2);
glutWireSphere(0.4, 20, 20);
glPopMatrix();

glPushMatrix();
glRotatef(satorb, 0.0, 0.0, 1.0);
glTranslatef(f6, 0.0, 0.0);
glRotatef(satrev, 0.0, 0.0, 1.0);
glColor3f(0.2, 0.1, 0.8);
glutWireSphere(0.2, 10, 10);
for (i = 0; i <= 100; i++);
{
        //glRotatef(satorb, 0.0, 0.0, 1.0);
        glTranslatef(0.2*.01*i, 0.0, 0.0);
        glRotatef(satrev * 12 * i, 0.0, 0.0, 1.0);
        if (i < 50)
                glColor3f(1, .5, 0);
        if (i >= 50)
                glColor3f(.5, .5, .5);

        glutSolidSphere(0.04, 10, 10);
        //glLoadIdentity();
        //glPopMatrix();
}

glTranslatef(0.3*.01, 0.0, 0.0);
glRotatef(satrev * 10, 0.0, 0.0, 1.0);
//if (i<50)
glColor3f(0, 1, 0);
//if (i >= 50)
// glColor3f(.5, .5, .5);

glutSolidSphere(0.04, 10, 10);
//glLoadIdentity();
glPopMatrix();





glPushMatrix();
glRotatef(uraorb, 0.0, 0.0, 1.0);
glTranslatef(f7, 0.0, 0.0);
glRotatef(urarev, 0.0, 0.0, 1.0);
glColor3f(0.0, 0.5, 1.0);
glutWireSphere(0.2, 10, 10);
glPopMatrix();

glPushMatrix();
glRotatef(neporb, 0.0, 0.0, 1.0);
glTranslatef(f8, 0.2, 0.0);
glRotatef(neprev, 0.0, 0.0, 1.0);
```

```
glColor3f(0, 0.1, 0.1);
glutWireSphere(0.1, 10, 10);
glPopMatrix();
```

CHAPTER 5

TESTING AND RESULTS

5.1 TEST CASES

In unit testing, the program modules that make up the system are tested individually. Unit testing focuses to locate errors in the working modules that are independent to each other. This enables to detect errors in coding and the logic within the module alone. This testing is also used to ensure the integrity of the data stored. The various routines were checked by passing the inputs and the corresponding output is tested. Test cases used in the project as follows:

Serial No	Metric	Description	Observation
1	**Keyboard Function**	Up arrow tilt up,down arrow side view. Q/q is for quit,f/F is for full screen.	Results obtained as expected
2	**Display Function**	Computers, send window, receive window, packets, timer are displayed.	Results obtained as expected
3	**Animation Effect**	Movement of planets and satillite movement.	Results obtained as expected

5.2 SCREENSHOTS

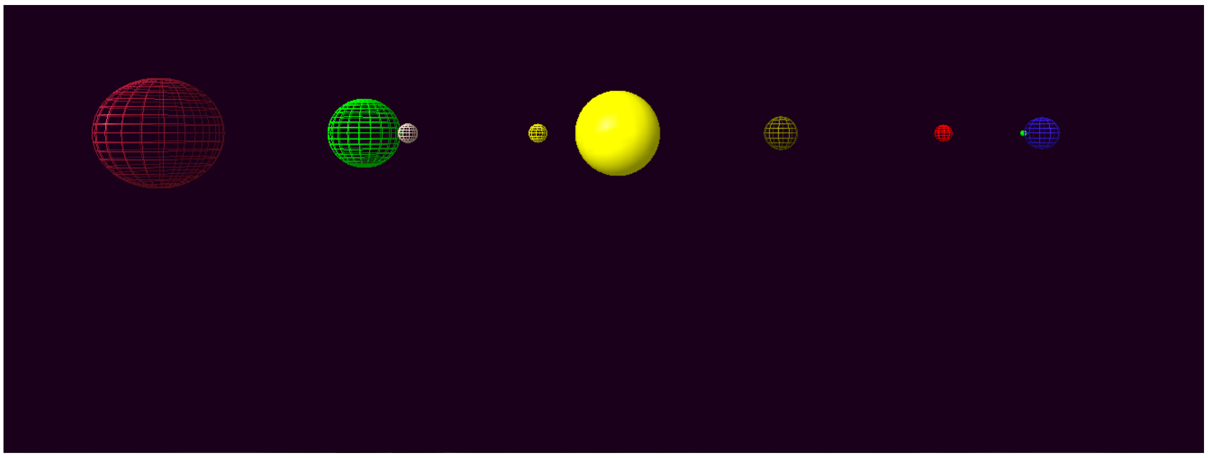

Fig .1 A Solar System

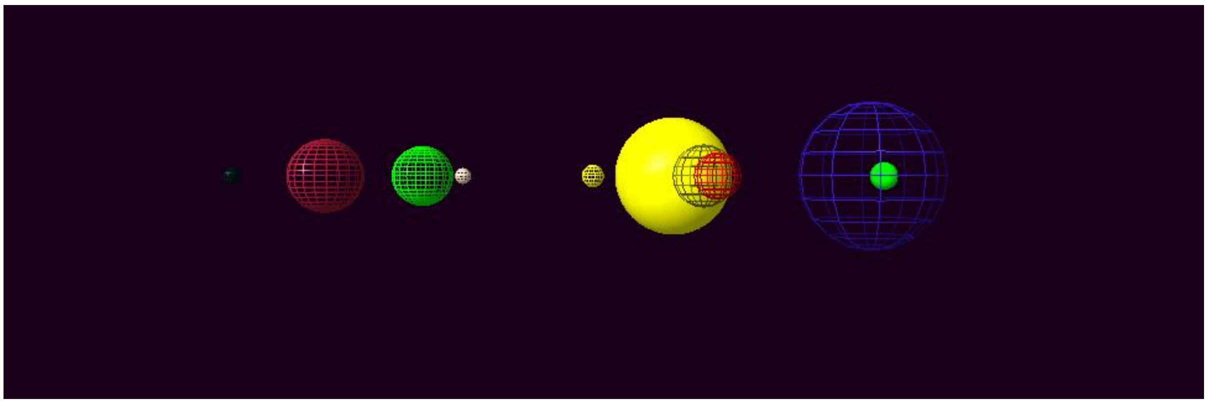

Fig.2 A Solar System

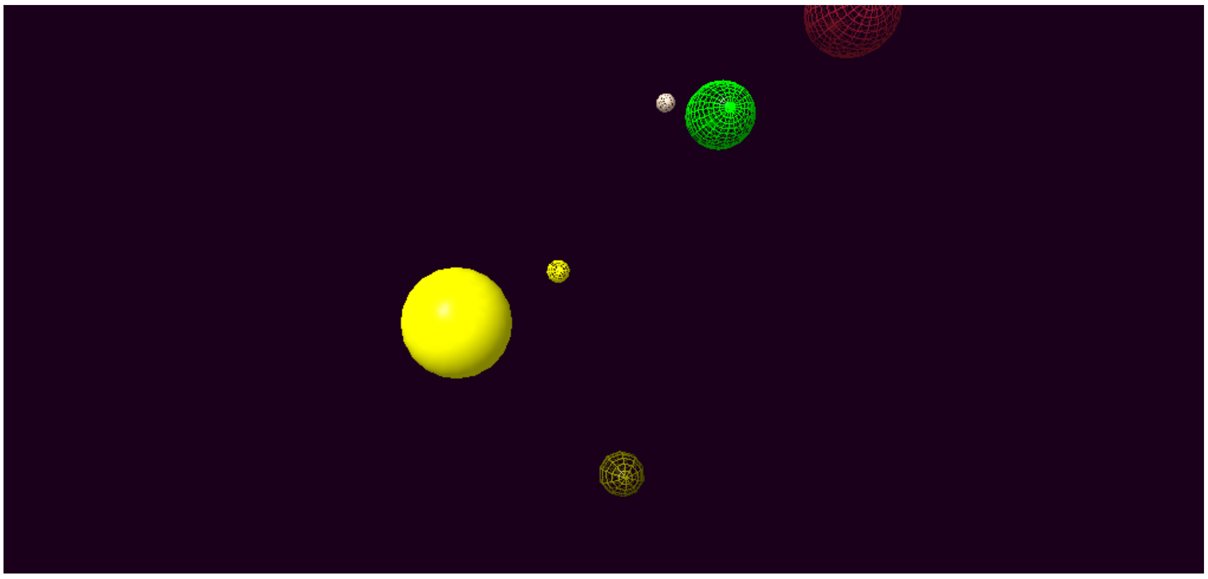

Fig.3 Top view of Solar System

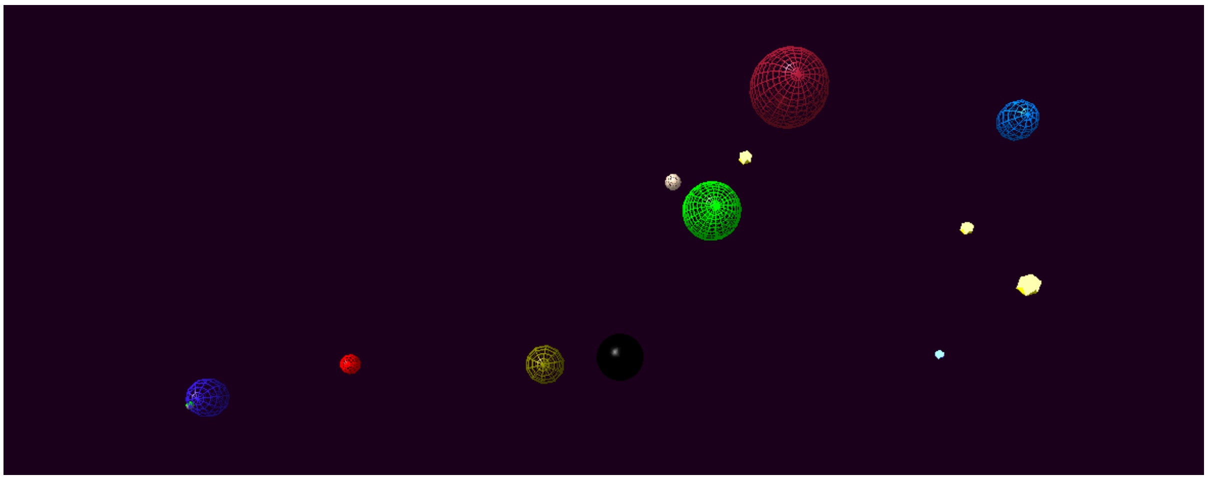

Fig.4 Black Hole appears

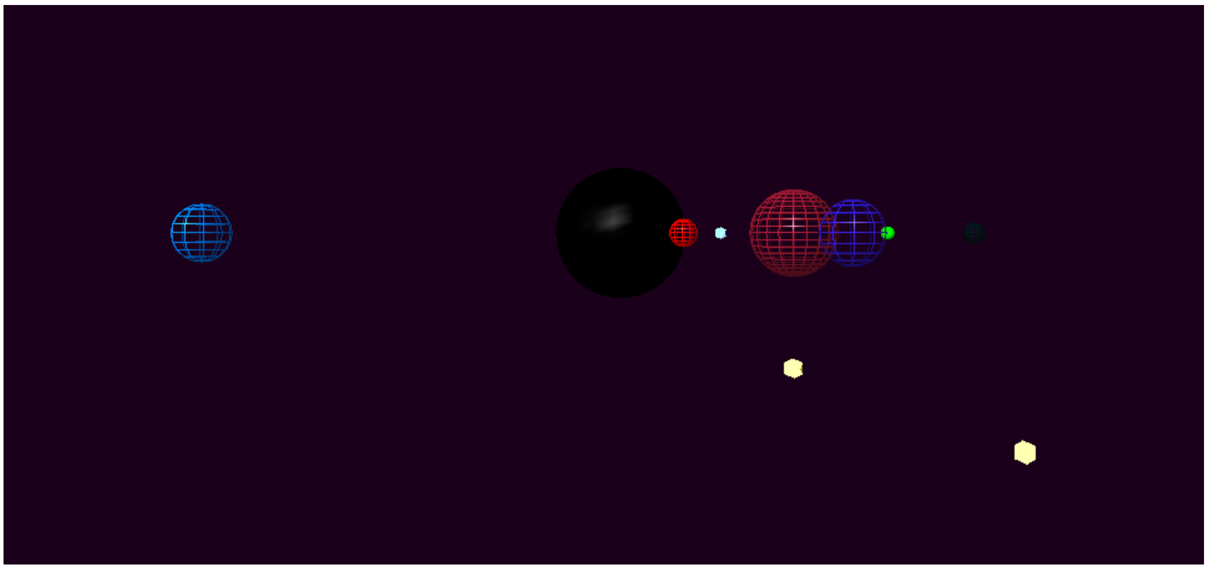

Fig.5 Black Hole engulfs all the planets

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENTS

The very purpose developing this project is to exploit the strength of Open GL graphics capability and to implement the concepts of simulation of black hole.
This project can be used by teachers as a way to help young students visualize a solar system and black hole. Or it can be used as a screen saver for laptop.

This package can be made more effectively for operators used in many computer languages like C,C++ etc .We can add more satellites and other objects in space can also be represented such as commets , dust particles etc.

REFERENCES

1. Edward Angel, “Interactive Computer Graphics A Top-Down Approach With OpenGL” 5th Edition, Addison-Wesley, 2008.

2. F.S. Hill, Jr., “Computer Graphics Using OpenGL”, 2nd Edition, Pearson Education, 2001.

3. JamesD.Foley, AndriesVan Dam, StevenK.Feiner, JohnF.Hughes, ”ComputerGraphes”, Second Edition, Addison-Wesley Proffesional, August 14,1995

4. www.opengl.org.

5. www.cs.uiowo.edu/~cwyman/classes/common

6. www.graphicsonline.com

7. www.stackoverflow.com