CHAPTER 1

**INTRODUCTION**

* 1. **Computer Graphics:-**

Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results. Computer graphics started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computers themselves. It has grown to include the creation, storage, and manipulation of models and images of objects. These models come from a diverse and expanding set of fields, and include physical, mathematical, engineering, architectural, and even conceptual structures, natural phenomena, and so on. Computer graphics today is largely interactive. The user controls the contents, structure, and appearance of the objects and of their displayed images by using input devices, such as keyboard, mouse, or touch-screen.

Due to close relationships between the input devices and the display, the handling of such devices is included in the study of computer graphics. The advantages of the interactive graphics are many in number. Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D patter-recognition abilities allow us to perceive and process data rapidly and efficiently. In many design, implementation, and construction processes today, the information pictures can give is virtually indispensable. Scientific visualization became an important field in the 1980s when the scientists and engineers realized that they could not interpret the prodigious quantities of data produced in supercomputer runs without summarizing the data and highlighting trends and phenomena in various kinds of graphical representations.

**1.2 OpenGL Interface:-**

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects.

Most of our applications will be designed to access OpenGL directly through functions in three libraries. They are:

**1. Main GL:**Library has names that begin with the letter gl and are stored in a library usually referred to as GL.

**2. OpenGL Utility Library (GLU):** This library uses only GL functions but contains code for creating common objects and simplifying viewing.

**3. OpenGL Utility Toolkit (GLUT):** This provides the minimum functionality that should be accepted in any modern windowing system.

**1.3 OpenGL Overview:-**

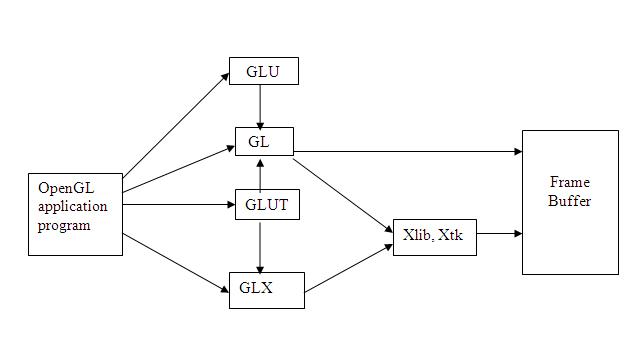
OpenGL(Open Graphics Library) is the interface between a graphic program and graphics hardware. It is streamlined. In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.

OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.

It is system-independent. It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.

**1.4 The openglinterface:-**

Our application will be designed to access OpenGL directly through functions in three libraries namely: GL, GLU, and glut.



Fig(1.1) The OpenGl Interface

CHAPTER 2

**LITERATURE SURVEY**

**2.1 The OpenGL Programming Guide –The Redbook**

The OpenGL graphics system is a software interface to graphics hardware. (The GL stands for Graphics Library.) It allows you to create interactive programs that produce color images of moving three-dimensional objects. With OpenGL, you can control computer-graphics technology to produce realistic pictures or ones that depart from reality in imaginative ways. This guide explains how to program with the OpenGL graphics system to deliver the visual effect you want.

**2.1.1 The OpenGL Programming Guide 5th Edition. The Official Guide to Learning OpenGL Version 2.1**

The OpenGL Programming Guide, Fifth Edition, provides definitive and comprehensive information on OpenGL Utility Library. The previous edition covered OpenGL through Version 1.3 and 1.4. This fifth edition of the best-selling “red-book” description the latest features of OpenGL Version 1.5 and 2.0, including the introduction of the OpenGL Shading language. You will find clear explanation of OpenGL functionality and many basic computer graphics techniques, such as building and rendering 3D models; interactively viewing texture mapping, antialiasing, fog and atmosphere effects, NURBS, image processing, and more.

**2.2 www.opengl.org for openGL tutorials**

**2.2.1 Previses offers consulting and software development for visualization leveraging OpenGL**

Previses is a new software development firm located in carry, NC that specializes in 3D visualization application. Recently completed and revived project include mash up visualization that create terrain visualization in 3D pdf and molecular modeling using OpenGL.

**2.2.2 VSFL-Very Simple Font Library**

Text rendering is very useful to display information on top of 3D world. VSFL aims at providing users with the ability to render bitmapped text in an OpenGL application using the core profile. With immediate mode gone in core OpenGL version, so are the vast majority of font libs that worked with OpenGL. Immediate mode was terribly slow, a code was very extensive. Vertex Buffer are clearly the way to go. This lib uses VAOs and vertex buffer to render text.

The basic functions like glColor3f (…); glRotatef (...), glTranslatef (...) etc that are most commonly used in the code are taken from the prescribed VTU Text book “INTERACTIVE COMPUTER GRAPHICS” 5th edition by Edward Angel.[1].

The lab programs in the syllabus also serve as a basic template for creating a project. The usage of colors and specifications are taken from the various programs that were taught in the lab.[1].

The VTU prescribed text book serves as a huge database of functions and they are used in the project.

CHAPTER 3

**REQUIREMENT SPECIFICATION**

The requirements can be broken down into 2 major categories namely hardware and software requirements. The formal specification the minimal hardware facilities expected in a system in which the project has to be run. The latter specifies the essential software needed to build and run the project.

**3.1 HARDWARE REQUIREMENTS:-**

* Dual Core Processor
* 2GB RAM
* 40GB Hard disk
* DVD drive
* Mouse and other pointing devices
* Keyboard

**3.2 SOFTWARE REQIREMENTS:-**

* Programming language – C/C++ using OpenGL
* Operating system – Linux operating system
* Compiler – C Compiler  Graphics library – GL/glut.h
* OpenGL 2.0

**3.3 FUNCTIONAL REQUIREMENTS:-**

**OpenGL APIs:**

If we want to have a control on the flow of program and if we want to interact with the window system thenwe use OpenGL API’S. Vertices are represented in the same manner internally, whether they are specified as two-dimensional or three dimensional entities, everything that we do are here will be equally valid in three dimensions. Although OpenGL is easy to learn, compared with other APIs, it is nevertheless powerful. It supports the simple three dimensional programs and also supports the advanced rendering techniques.

**GL/glut.h:**

We use a readily available library called the OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system. The application program uses only GLUT functions and can be recompiled with the GLUT library for other window system. OpenGL makes a heavy use of macros to increase code readability and avoid the use of magic numbers.

CHAPTER 4

**ABOUT THE PROJECT**

**4.1 Overview:-**

My project is a simple mountain view consists of sun and mountain. The sun will move from one end to other end. And the mountains are static in its position. The sun starts moving upward from one end of the mountain to the top of the sky and it return back in downward position and move to other end of the mountain. While moving from one end to other end the color of sun turns out from light shade to dark shade. This project containsa background which turns its color from blue to black when sun rise and set down.

**4.2 User interface:-**

The interface is mainly concentrated on use ofkeyboard. Clicking left key and right key will help us to move the position of sun. By pressing key ‘**left**’ the sun move towards left. If we press the key ‘**right**’the sun move towards right.

**4.3 Purpose:-**

The aim of this project is to develop a graphics package which supports basic operations which include motion of sun using Open GL. The package must also has a user-friendly interface. The objective of developing this model was to design and apply the skills we learnt in class.

CHAPTER 5

**IMPLEMENTATION**

**5.1 Functions in open gl:-**

* **void glClear(glEnum mode);**

Clears the buffers namely color buffer and depth buffer. mode refers to GL\_COLOR\_BUFFER\_BIT or DEPTH\_BUFFER\_BIT.

* **void glTranslate[fd](TYPE x, TYPE y, TYPE z);**

Alters the current matrix by displacement of (x, y, z), TYPE is either GLfloat or GLdouble.

* **void glutSwapBuffers();**

Swaps the front and back buffers.

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

Specifies which matrix will be affected by subsequent transformations, Mode can be GL\_MODELVIEW or GL\_PROJECTION.

* **glViewport;**

glViewport specifies the affine transformation of x and y from normalized device coordinates to window coordinates. Let x nd y nd be normalized device coordinates. Then the window coordinates x w y w are computed as follows:

x w = x nd + 1 ? width 2 + x

y w = y nd + 1 ? height 2 + y

Viewport width and height are silently clamped to a range that depends on the implementation.

* **void glLoadIdentity( );**

Replace the current matrix with the identity matrix.glMatrixMode sets the current matrix mode. mode can assume one of three values:

GL\_MODELVIEW

Applies subsequent matrix operations to the modelview matrix stack.

GL\_PROJECTION

Applies subsequent matrix operations to the projection matrix stack.

GL\_TEXTURE

Applies subsequent matrix operations to the texture matrix stack.

* **void glEnable(GLenum feature);**

Enables an OpenGL feature. Feature can be GL\_DEPTH\_TEST (enables depth test for hidden surface removal), GL\_LIGHTING (enables for lighting calculations), GL\_LIGHTi (used to turn on the light for number i), etc.

* **void glPushMatrix();**
* **void glPopMatrix();**

There is a stack of matrices for each of the matrix modes. In GL\_MODELVIEW mode, the stack depth is at least 32. In the other modes, GL\_COLOR, GL\_PROJECTION, and GL\_TEXTURE, the depth is at least 2. The current matrix in any mode is the matrix on the top of the stack for that mode.glPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on top of the stack is identical to the one below it.glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack.Initially, each of the stacks contains one matrix, an identity matrix.It is an error to push a full matrix stack or to pop a matrix stack that contains only a single matrix. In either case, the error flag is set and no other change is made to GL state.

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

Initializes GLUT; the arguments from main are passed in and can be used by the application.

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

Requests a display with the properties in the mode; the value of mode is the logical OR of options including the color model (GLUT\_RGB, GLUT\_INDEX) and buffering (GLUT\_SINGLE, GLUT\_DOUBLE).

* **void glutCreateWindow(char \*title);**

Creates a window on display; the string title can be used to label the window. return value provides a reference to the window that can be used when there are multiple windows.

* **void glutMainLoop();**

Causes the program to enter an event-processing loop.

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

Registers the display function func that is executed when the window needs to be redrawn.

* **void glutKeyboardFunc(void \*f(char key, int width, int height))**

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

* **void glClearColor(GLclampf r, GLclampf g, GLclamp b, Glclamp a)**

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

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

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

* **void glColor3[b I f d ub us ui](TYPE r, TYPE g, TYPE b)**

Sets the present RGB colors. Valid types are bytes(b), int(i), float(f), double(d), unsigned byte(ub), unsigned short(us), and unsigned int(ui). The maximum and minimum value for floating point types are 1.0 and 0.0 respectively, whereas the maximum and minimum values of the discrete types are those of the type, for eg, 255 and 0 for unsigned bytes.

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

Specifies the initial height and width of the window in pixels.

* **void glutReshapeFunc(void \*f(int width, int height));**

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

* **5.2 User defined functions:-**
* **Void drawballs():-**

It is used to draw the ball which acts as a sun. This function is also used to set the color to the ball.

* **Void drawAv():-**

This function is used to create the mountain.

* **Void drawScene():-**

This function is used to create the scene by combining mountain function and the sun function.

* **Void update():-**

This function used to update the values of sun for every 15 millisecond.

* **Int main():-**

The main function is used for creating the window for display of the model of the atom. Here, we create menu for ease of use for the user. The callback functions, i.e., mouse callback, keyboard callback, display callback, idle callback, are written in main. The callback functions registered in main ( ) are,

glutKeyboardFunc (mykeys);

glutDisplayFunc (display);

glutReshapeFunc (reshape);

glutMouseFunc (mouse);

* **Source code used in implementation of the program:-**

#include<GL/glut.h>

floatballX = -0.8f;

floatballY = -0.3f; floatballZ = -1.2f;

floatcolR=3.0; floatcolG=1.5;

floatcolB=1.0; floatbgColR=0.0;

floatbgColG=0.0; floatbgColB=0.0;

staticint flag=1;

voiddrawBall(void)

{

glColor3f(colR,colG,colB); //set ball colour

glTranslatef(ballX,ballY,ballZ); //moving it toward the screen a bit on creation

glutSolidSphere (0.05, 30, 30); //create ball.

}

voiddrawAv(void)

{

glBegin(GL\_POLYGON);

glColor3f(1.0,1.0,1.0);

glVertex3f(-0.9,-0.7,-1.0);

glVertex3f(-0.5,-0.1,-1.0);

glVertex3f(-0.2,-1.0,-1.0);

glVertex3f(0.5,0.0,-1.0);

glVertex3f(0.6,-0.2,-1.0);

glVertex3f(0.9,-0.7,-1.0);

glEnd();

}

voidkeyPress(int key, int x, int y)

{

if(key==GLUT\_KEY\_RIGHT) ballX += 0.05f;

if(key==GLUT\_KEY\_LEFT) ballX -= 0.05f;

glutPostRedisplay();

}

voidinitRendering()

{

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);//Enable lighting

glEnable(GL\_LIGHT0); //Enable light #0

glEnable(GL\_LIGHT1); //Enable light #1

glEnable(GL\_NORMALIZE); //Automatically normalize normals

//glShadeModel(GL\_SMOOTH); //Enable smooth shading

} //Called when the window is resized

voidhandleResize(int w, int h)

{

//Tell OpenGL how to convert from coordinates to pixel values

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION); //Switch to setting the camera perspective //Set the camera perspective

glLoadIdentity(); //Reset the camera

gluPerspective(45.0,(double)w / (double)h,1.0,200.0);

//The far z clipping coordinate

}

voiddrawScene()

{

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

glClearColor(bgColR,bgColG,bgColB,0.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity(); //Add ambient light

GLfloatambientColor[] = {0.2f, 0.2f, 0.2f, 1.0f}; //Color (0.2, 0.2, 0.2)

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambientColor); //Add positioned light

GLfloatlightColor0[] = {0.5f, 0.5f, 0.5f, 1.0f}; //Color (0.5, 0.5, 0.5)

GLfloatlightPos0[] = {4.0f, 0.0f, 8.0f, 1.0f}; //Positioned at (4, 0, 8)

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightColor0);

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPos0); //Add directed light

GLfloatlightColor1[] = {0.5f, 0.2f, 0.2f, 1.0f}; //Color (0.5, 0.2, 0.2) //Coming from the direction (-1, 0.5, 0.5)

GLfloatlightPos1[] = {-1.0f, 0.5f, 0.5f, 0.0f};

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, lightColor1);

glLightfv(GL\_LIGHT1, GL\_POSITION, lightPos1); //drawing the SUN

glPushMatrix();

drawBall();

glPopMatrix(); //drawing the Mount Avarest

glPushMatrix();

drawAv();

glPopMatrix();

glutSwapBuffers();

} //float \_angle = 30.0f;

void update(int value)

{

if(ballX>0.9f)

{

ballX = -0.8f;

ballY = -0.3f;

flag=1; colR=2.0;

colG=1.50; colB=1.0;

bgColB=0.0;

}

if(flag)

{

ballX += 0.001f;

ballY +=0.0007f;

colR-=0.001; //colG+=0.002;

colB+=0.005;

bgColB+=0.001;

if(ballX>0.01)

{

flag=0;

}

} if (!flag)

{

ballX += 0.001f;

ballY -=0.0007f;

colR+=0.001;

colB-=0.01;

bgColB-=0.001;

if(ballX<-0.3)

{

flag=1;

}

} glutPostRedisplay();

//Tell GLUT that the display has changed //Tell GLUT to call update again in 25 milliseconds

glutTimerFunc(15, update, 0);

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(900,900);

glutCreateWindow("Sun");

initRendering();

glutDisplayFunc(drawScene);

glutSpecialFunc(keyPress);

glutReshapeFunc(handleResize);

glutTimerFunc(15, update, 0);

glutMainLoop();

return(0);

}

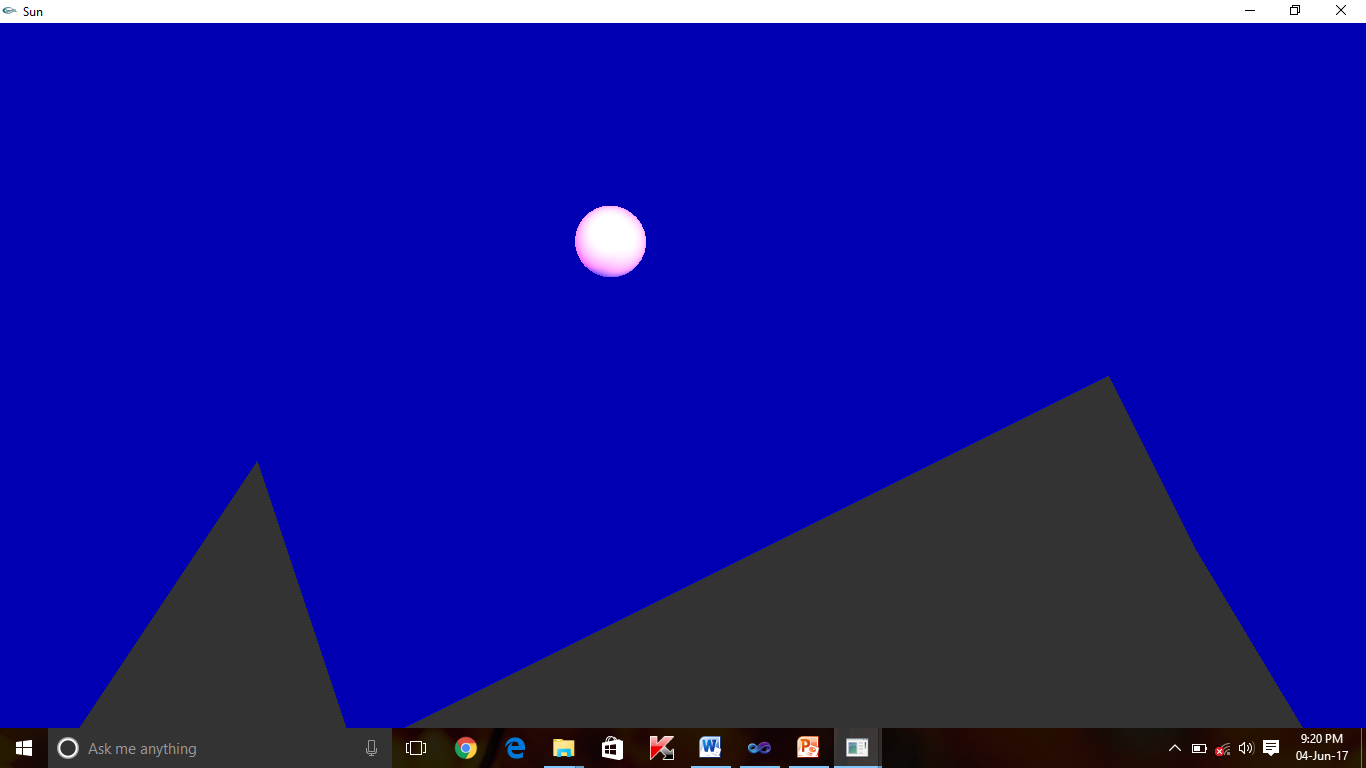
CHAPTER 6

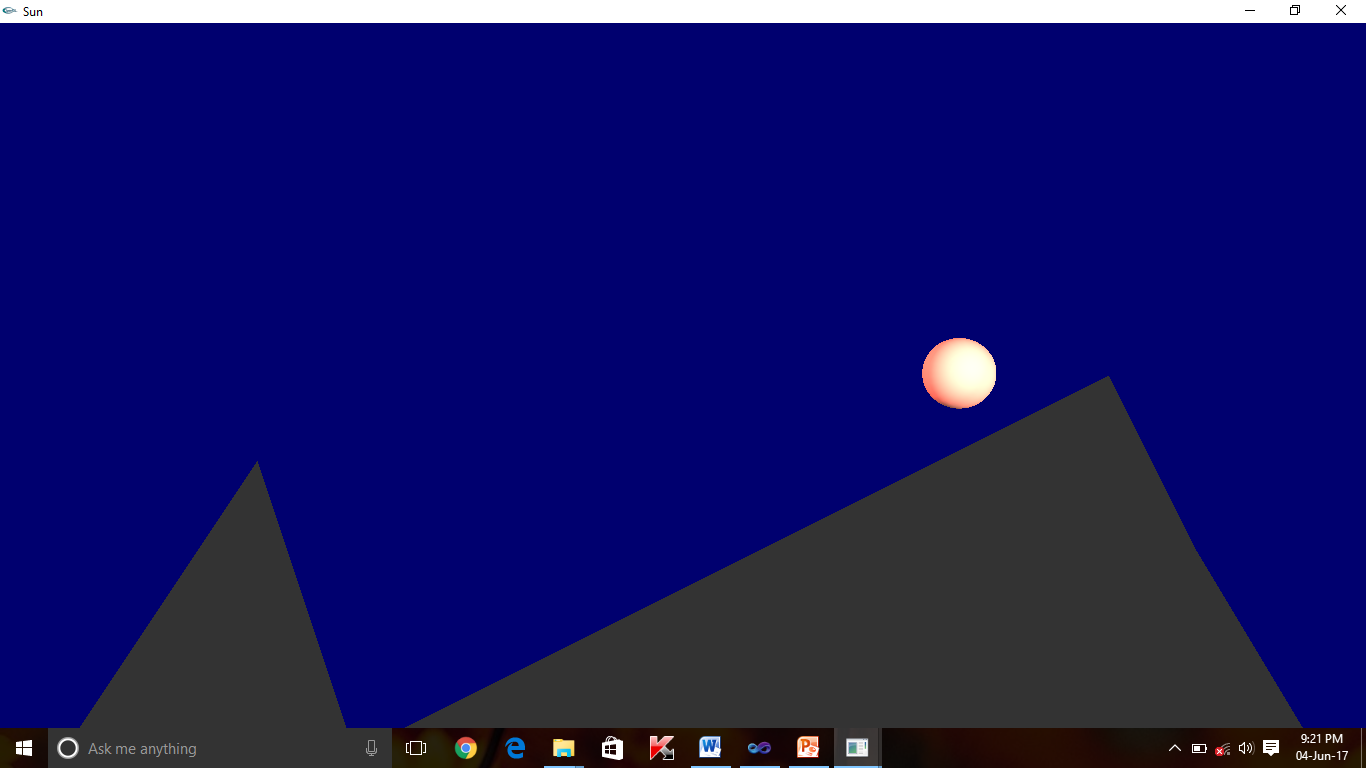
**SYSTEMTESTING**

Testing process started with the testing of individual program units such as functions or objects. These were then integrated into sub-systems and systems, and interactions of these units were tested. Testing involves verification and validation. Validation: “Are we building right product?” Verification: “Are we building the product right?” The ultimate goal of the verification and validation process is to establish confidence that the software system is ‘fit for purpose’. The level of required confidence depends on the system’s purpose, the expectations of the system users and the current marketing environment for the system. With the verification and validation process, there are two complementary approaches to the system checking and analysis: Software inspections or peer reviews analyses and check system representations such as the requirements document, design diagrams, and the program source code. Software testing involves running an implementation of the software with test data.

**OUTPUT SCREENS**









**CONCLUSION**

The development of computer graphicsmini project of sun rise and set with required functionalities like moving sun, and mountains had done successfully. Also there is implementation of user interactive computer graphics with keyboard.

We started with modest aim with no prior experience in any programming project as this, but ended up in learning many things, fine tuning the programming skill and getting into the real world of software development with an exposure to corporate environment. During the development of any software of significant utility, we are faced with the trade-off between speed of execution and amount of memory consumed. This is simple interactive application. It has open source code and no security features has been included. The user is free to alter the code for feature enhancement. Verification of all possible types of function are taken care. Care was taken to avoid bugs. Bugs may be reported to creator as the need may be. So, we conclude on note that we are looking forward to develop more such project with an appetite to learn more in computer graphics.

**BIBLIOGRAPHY**

[1].Interactive computer Graphics: A Top-Down Approach with OpenGL-Edward Angel, 5th edition, Addison-Wesllesley,2008

[2] Online tutorials for OpenGL preparation on project Steam engine.

[3]OpenGL Redbook and Bluebook for reference.

[4][www.opengl.org](http://www.opengl.org/) for OpenGL tutorials.

[5][www.stackoverflow.com](http://www.stackoverflow.com/)

[6][www.sourcecode.com](http://www.sourcecode.com/)