CHAPTER 1

INTRODUCTION

1.1 Introduction to Computer Graphics

COMPUTER GRAPHICS is concerned with all aspects of producing pictures or images using a computer. The field began humbly almost 50 years ago, with the display of a few lines on a cathode-ray tube (CRT); now, we can create images by a computer that are indistinguishable from photographs of real objects. We routinely train pilots with simulated airplanes, generating graphical displays of a virtual environment in real-time. Feature-length movies made entirely by computer have been successful, both critically and financially. Massive multiplayer games can involve tens of thousands of concurrent participants.

VISUALIZATION is any technique for creating images, diagrams, or animations to communicate a message.

1.2 Image Types

➢ 2D computer graphics:

2D computer graphics are the computer-based generation of digital images mostly from two-dimensional models, such as 2D geometric models, text, and digital images, and by techniques specific to them.2D computer graphics are mainly used in applications that were originally developed upon traditional printing and drawing technologies, such as typography, cartography, technical drawing, and advertising. Two-dimensional models are preferred because they give more direct control of the image than 3D computer graphics, whose approach is more akin to photography than to typography.

There are two approaches to 2D graphics: vector and raster graphics.

- Pixel art: Pixel art is a form of digital art, created through the use of raster graphics software, where images are edited on the pixel level.
- Vector graphics: Vector graphics formats are complementary to raster graphics, which is the representation of images as an array of pixels, as it is typically used for the representation of photographic images.

➢ 3D computer graphics:

With the birth of the workstation computers(like LISP machines, paint box computers and Silicon Graphics workstations)came the 3D computer graphics.3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes0 of performing calculations and rendering 2Dimages.

1.3 Applications of Computer Graphics

Some of the applications of computer graphics are listed below:

- Computational biology
- Computational physics
- Computer-aided design
- Computer simulation
- Digital art
- Education
- Graphic design
- Video Games
- Virtual reality
- Web design

1.4 Introduction to OpenGL

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.

Basic OpenGL Operation

The figure shown below gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

1.5 Introduction to GLUT

GLUT is the OpenGL Utility Toolkit, a Windows system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works on both Win32 PCs and X11 workstations.

GLUT is designed for constructing small to medium-sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits like Motif. GLUT is simple, easy, and small. My intent is tokeep GLUT that way.

The GLUT library supports the following functionality:

- Multiple windows for OpenGL rendering.
- Call back-driven event processing.
- An `idle' routine and timers.
- Utility routines to generate various solid and wireframe objects.
- Support for bitmap and stroke fonts.
- Miscellaneous window management functions.

1.6 Overview of the project

There are many phase of moon , it is define as - The lunar phase or phase of the moon is the shape of the illuminated portion of the Moon as seen by an observer, usually on Earth.

Today for VTU 6th sem mini project in the computer graphics lab we are going to have the phases of moon. In the this program we showcase the camera orbits a lit moon to simulate the phases of the moon. This program used the OOP used in C++, the use of the classes and the methods.

1.7 Aim of the project

The main aim of this project is to visually present the users with the phase of moon. This project gives a better understanding of the different phase of moon.

CHAPTER 2

SOFTWARE REQUIREMENT SPECIFICATION

2.1 Software Requirements

- Operating System: Windows 98/XP or Higher
- Programming Language: C/C++
- IDE : CodeBlocks/Dev-C++

2.2 Hardware Requirements

This package has been developed on:

- Processor: Pentium Processor
- Processor Speed: 333 MHz
- RAM: 32 MB or Higher
- Graphics Card: 512MB
- Monitor: Color
- Keyboard: Low Profile, Dispatch able Type
- I/O Parts: Mouse, Monitor

CHAPTER 3

DESIGN

3.1 Implementation

Input: The user gives input through the mouse buttons.

Output: Different phase of moon are selected based on the input.

Step 1: Open and run the project.

Step 2: Record Input from the user. (Left mouse click or Right mouse click)

Step 3: If the input is left mouse click then:

3a: Phase of the moon is advanced.

Step 5: Stop.

3.2 Working

First we create a class for the moon. In the class moon we create an OpenGL display list, which is created with create(), and the draw() method calls it.We have use the sphere to feature the moon, which has radius 1 centered at the origin. Another class which we have created is orbiter, this is an object that flies on a circle of a certain radius on the xz plane. The radius to it is supplied at the time construction. Since there is one one moon so it must be static, hence we declared the moon to be static variable.

How we are showing the phase? It's easy by using the lightning property of OpenGL. We have used the GL_LIGHT0 to focus the light on the moon, making the all phase visible as it go. Pointing in the direction of vertex 1,1,1 it's is possible to show all the moon phases.

CHAPTER 4

IMPLEMENTATION

4.1 OpenGL Functions

The various functions used in implementing this project are:

- **Glut:** An introduction to the OpenGL Utility Toolkit.
- **GlutInit( )** : Initialize the GLUT library and graphics system.

  Declaration: void GlutInit (int argc, char **argv);

  Description: To start the graphics system, you must first call GlutInit ( ), GlutInit will initialize the GLUT library and negotiate a session with the window system. During this process, GlutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.
- **glutInitDisplayMode( ) :** Sets the initial display mode.

  Declaration: void glutInitDisplayMode(unsigned int mode);

  Description: The initial display mode is used when creating top-level windows, sub windows, and overlays to determine the OpenGLdisplay mode for the to-be-created window or overlay.

- **glutDisplayFunc:** Register the display function that is executed when the window needs to be redrawn.
- **glutPostRedisplay( ):** Request that the display callback be executed after the current callback returns.
- **glutMainLoop( ):** Cause the program to enter an event processing loop. It should be the last statement in main( ).
- **glPushMatrix( ) and glPopMatrix( ):** Pushes to and pops from the matrix stack corresponding to the current matrix mode.
- **glBegin( ):** Initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL_POINTS, GL_LINES, and GL_POLYGON.

- **glEnd( ):** Terminates a list of vertices .
- **glutReshapeFunc( )** : Sets the reshape callback for the current window.

- **glutCreateWindow ( ):** Creates a top-level window.
- **glutInitWindowSize ( ):** Requests future windows to open at a given width/height.
- **glOrtho( ):**

  Declaration: void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far).

  Description: It defines an orthographic viewing volume with all parameters measured from the center of the projection plane.
- **glViewport( ):** Set the viewport.
- **glLoadIdentity ( ):** Replace the current matrix with the identity matrix.
- **void menu(int item):** This function is used to perform menu options.
- **int main(int argc, char *argv[]):** The program starts its execution from this function. And this function calls all the user-defined and graphical functions.

4.2 Source Code

```
// In this program the camera orbits a lit moon to simulate the phases of the
// moon.

#ifdef __APPLE_CC__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include <cmath>

// A class for the moon.  A moon is really just an OpenGL display list.  The
// display list is created with create(), and the draw() method calls it.
// The reason we don't create the display list in the constructor is that
// clients may want to declare moon objects before the call to initializing
// OpenGL.
//
// The moon is a sphere of radius 1 centered at the origin, built from 25
// slices and 25 stacks, lit with GL_LIGHT0 as a directional light pointing
// along <1,1,1>.
class Moon {
  int displayListId;
public:
  void create() {
    displayListId = glGenLists(1);
    glNewList(displayListId, GL_COMPILE);
    GLfloat direction[] = {-1.0, -1.0, -1.0, 0.0};
    glLightfv(GL_LIGHT0, GL_POSITION, direction);
    glutSolidSphere(1.0, 25, 25);
    glEndList();
```

```
  }
  void draw() {
    glCallList(displayListId);
  }
};

// The one and only moon.
static Moon moon;

// An orbiter is an object that flies on a circle of a certain radius on the
// xz plane.  You supply the radius at construction time.
class Orbiter {
  double radius;
  double u;
public:
  Orbiter(double radius): radius(radius), u(0.0) {}
  void advance(double delta) {u += delta;}
  void getPosition(double& x, double& y, double& z) {
    x = radius * cos(u);
    y = 0;
    z = radius * sin(u);
  }
};

// The one and only orbiter.
static Orbiter orbiter(5.0);

// Clears the window (and the depth buffer) and draws the moon as viewed from
// the current position of the orbiter.
void display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
```

```
  glPushMatrix();
  glLoadIdentity();
  double x, y, z;
  orbiter.getPosition(x, y, z);
  gluLookAt(x, y, z, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  moon.draw();
  glPopMatrix();
  glutSwapBuffers();
}

// Advances the orbiter and requests to draw the next frame.
void timer() {
  orbiter.advance(0.1);
  glutPostRedisplay();
 //glutTimerFunc(1000/60, timer, v);
}

void mouse(int button ,int state, int x, int y){
    if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        timer();
}

// reshape() fixes up the projection matrix so that we always see a sphere
// instead of an ellipsoid.
void reshape(GLint w, GLint h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0, GLfloat(w) / GLfloat(h), 1.0, 10.0);
}
```

```
// Enables depth testing, enables lighting for a bright yellowish diffuse
// light, and creates a moon.
void init() {
  glEnable(GL_DEPTH_TEST);
  GLfloat yellow[] = {1.0, 1.0, 0.5, 1.0};
  glLightfv(GL_LIGHT0, GL_DIFFUSE, yellow);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  moon.create();
}

// The usual application code.
int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowPosition(80, 80);
  glutInitWindowSize(500, 500);
  glutCreateWindow("The Moon");
  glutDisplayFunc(display);
  glutMouseFunc(mouse);
  //glutTimerFunc(100, timer, 0);
  glutReshapeFunc(reshape);
  init();
  glutMainLoop();
}
```

CHAPTER 5 5.1 Results

Following figures are different phases of Moon

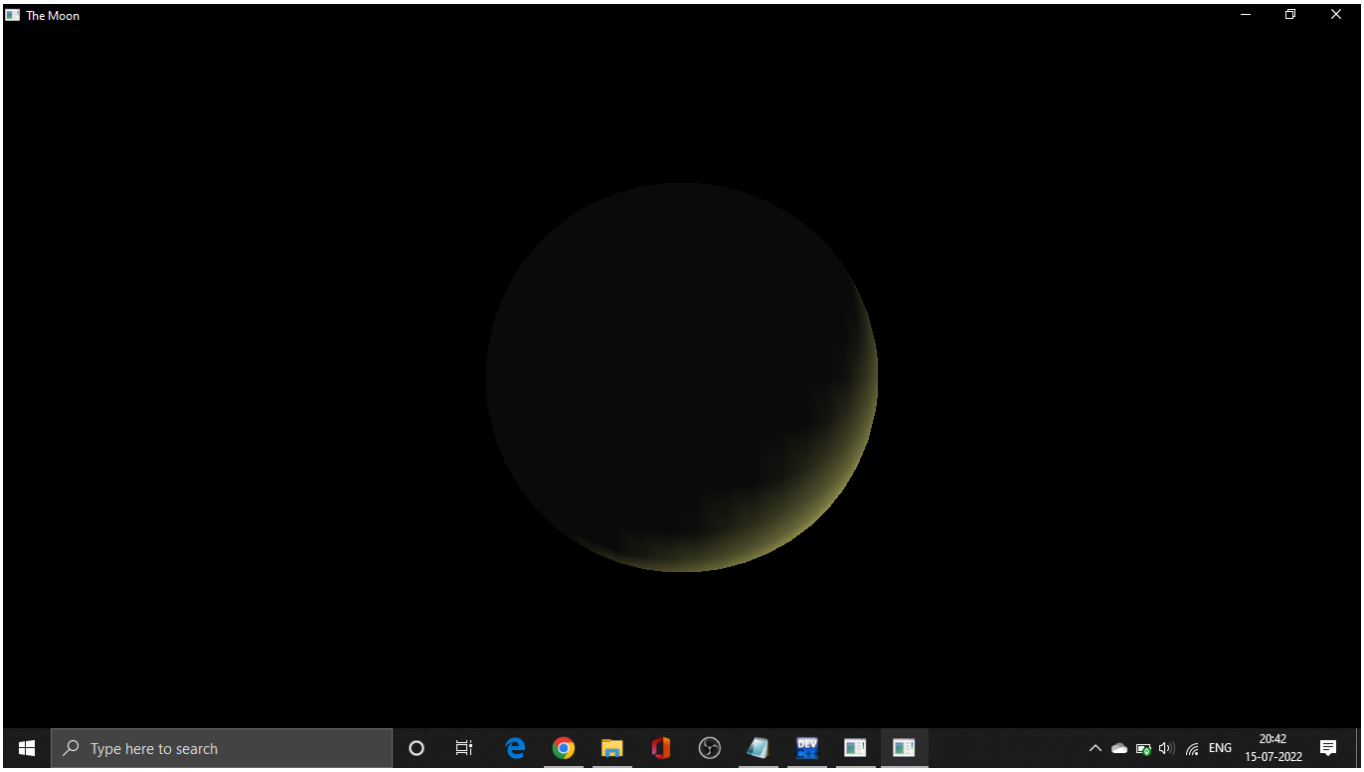

Figure 5.1

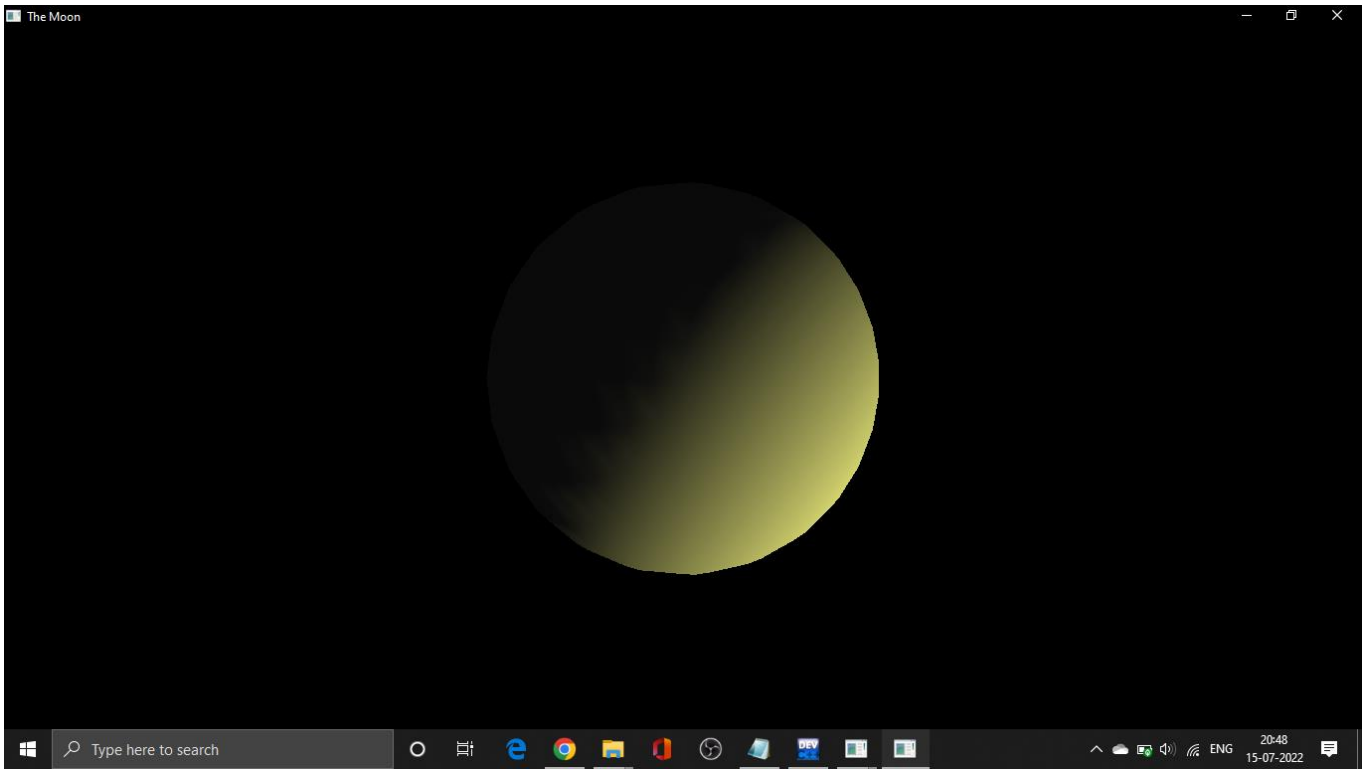

Figure 5.2

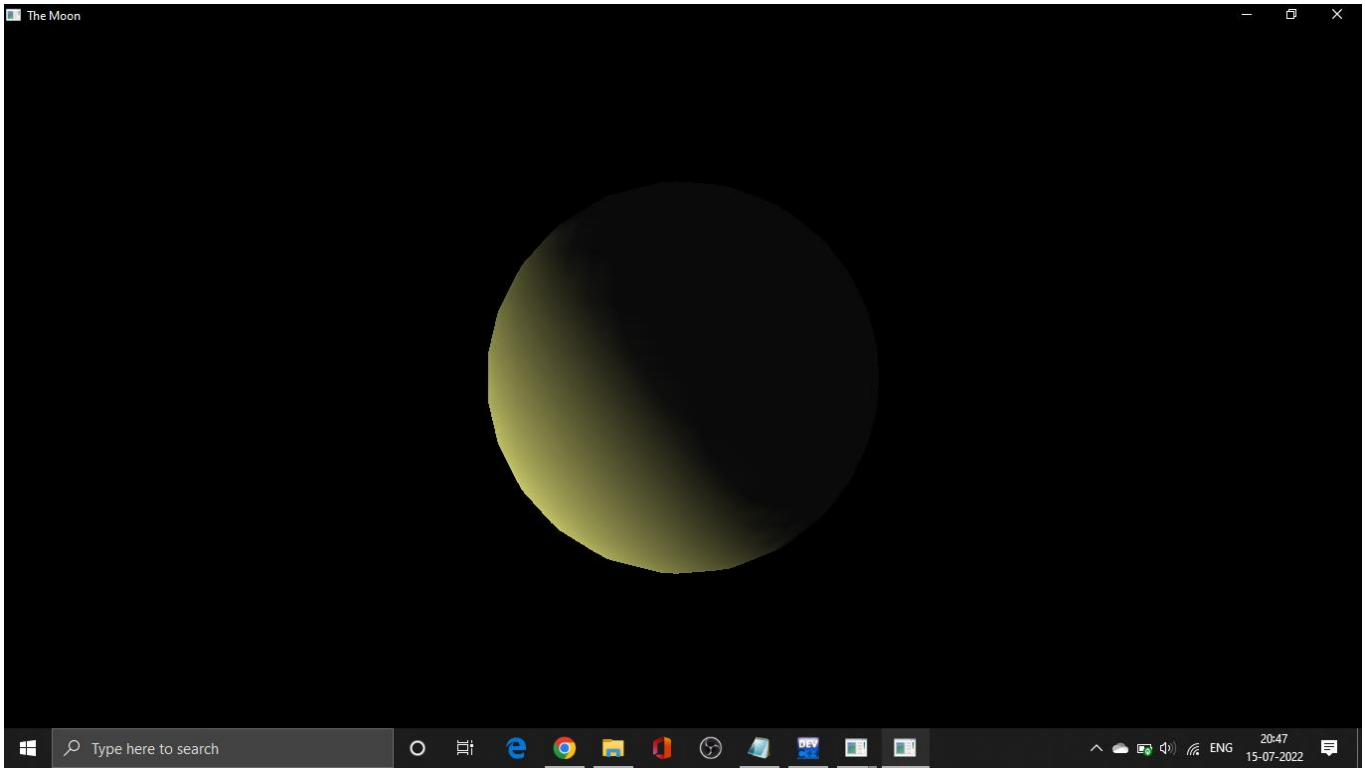

Figure 5.3

Conclusion

This project helps to understand the concepts behind OpenGL and its programming. This project gives a brief insight as to how programs, involving graphics, are written using OpenGL.

This project demonstrates how the OpenGL can be used to implement graphics which can be applied in various fields such as advertising industries to endorse company's products animation studios to make animated films, cartoons, gaming industries by displaying the use of high-end graphics in designing games, in engineering, architecture, medical fields by creating real-world models for better understanding, clarity and bringing out fresh, new ideas to enhance them.

The conclusion from this project is that it describes phase of moon clearly so that the users can understand them without any doubts.

References

[1]. Edward Angel *"Interactive Computer Graphics"* Pearson Education, 5th Edition.2008
[2]. Donald Hearn & Pauline baker *"Computer Graphics with OpenGL"* 3rd Edition. 2011
[3].www.opengl.org
[4]. www.freeglut.sourcefourge.net
[5]. www.github.com
[6].www.openglcg.blogspot.in