VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELGAVI-590018

Computer Graphics Laboratory
A Mini-Project Report
On

"SOLAR SYSTEM"

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*

Submitted by:

NAMRATHA H V 1RN18CS067
SAHANA S GOWDA 1RN18CS089

Under the Guidance of:

Mrs. S Mamatha Jajur
Asst. Professor
Dept. of CSE

Mrs. Swathi G
Asst. Professor
Dept. of CSE

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

2020-2021

RNS Institute of Technology

Channasandra, Dr.Vishnuvardhan Road,
Bengaluru-560 098

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

Certified that the mini project work entitled **"Solar System"** has been successfully carried out by **Namratha H V** bearing USN **1RN18CS067** and **Sahana S Gowda** bearing USN **1RN18CS089**, 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 2020-21. 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
Asst Professor
Dept. of CSE

Dr. Kiran P
Prof. and Head
Dept of CSE

External Viva:

Name of the Examiners **Signature with Date:**

1.

2.

ACKNOWLEDGEMENT

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

We would like to thank **Dr. Kiran P**, 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, Asst Prof, Department of CSE, RNSIT, Bangalore** and **Mrs. Swathi G**, **Asst 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:16/08/2021 **Namratha H V** **1RN18CS067**

Place: Bengaluru **Sahana S Gowda** **1RN18CS089**

ABSTRACT

The project “SOLAR SYSTEM” is meant as a source of recreation where one can sit in front of the computer and have the vision of a plant in space. This package is developed to provide opportunities to climb aboard the earth for the adventure of the lifetime. It is aimed to create sun and planets and give constant motion to these objects. The lighting effect in the background appears as though the planet is rotating and revolving around the sun in the galaxy. The most important aspect of this project is that one can sit back, relax, and watch constantly occurring motion of the planet and the stars just depicting the fact that “as passengers of the earth our voyage never ends!”

The Solar System consists of the Sun and the astronomical objects bound to it by gravity, all of which formed from the collapse of a giant molecular cloud approximately 4.6 billion years ago. Of the many objects that orbit the Sun, most of the mass is contained within eight relatively solitary planets whose orbits are almost circular and lie within a nearly flat disc called the ecliptic plane. The four smaller inner planets, Mercury, Venus, Earth and Mars, also called the terrestrial planets, are primarily composed of rock and metal. The four outer planets, the gas giants, are substantially more massive than the terrestrials. The two largest, Jupiter and Saturn, are composed mainly of hydrogen and helium; the two outermost planets, Uranus and Neptune, are composed largely of ices, such as water, ammonia and methane, and are often referred to separately as "ice giants".

The aim of this project is to show the shadow implementation using OPENGL which include Texture, Light properties also transformation operations like translation, rotation, scaling etc. on objects and also Sound effects. The package must also have a user-friendly interface.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

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 illustrates a simple solar system in a 3D space where the planets will have retrograde motion towards the sun. In this project, there are 2 types of models that will be used to illustrate the planet. The first model is a simple sphere which represents the 3D shape for most of the planets. The second model is a sphere with a circular ring surrounding the object. This model is used on planets such as Saturn.

For the object or model texture, the images are taken from the internet. The final 3D object is the end product of a modified 3D object that is extracted from the internet.

Fig: 1.1 3D Object

For the 3D objects, Blender Software used as the tool to modify some of the mesh shapes. For the rest of the project to achieve the end result, all the method is built by using the C++ programming language and various computer graphic libraries such as OpenGL.

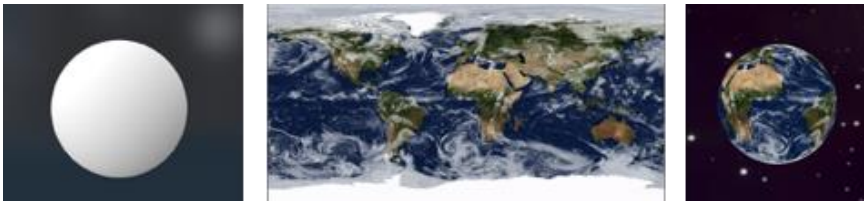

Fig: 1.2 3D Object texture mapping

To have the texture wrapped in the object, first, the texture image must be extracted using readBMPFile function. Next, the image can be assigned to a particular model or object. For the solar system project, the texture mapping process is quite easy because the texture image will only wrap the sphere object. There are several custom parameters that need to be assigned to the texture configuration. Two texture configuration that we changed are the configuration for the wrapping and for the filtering. Wrapping configuration is related to how the texture image will be displayed when the coordinates are outside of the image range. Filtering configuration is related to how the computer graphics handle the pixels when the image is being scaled or stretched.

There are two animations that affect the planets position and rotation angle. The first animation is a simple planet rotation in prograde motion. To find the rotation angle, a constant value is multiplied with the increasing index of the frame. The constant value also acted as the speed movement of the rotation. Thus, if the rotation speed needs to be faster, the constant value can be updated to a bigger float value. The second animation is the orbital motion that affects the planet positions in the space. The motion is particularly called the retrograde motion of the planet in the direction of the sun. On the real solar system, it'll create a skewed orientation. The planet can be translated/updated to the directed position in space/window. Both animations will update the planet position and rotation value as the frame index increases.

We assigned some of the keyboard function to control the camera. In this case, the camera will follow the position assigned to the camera object. The basic controls are the UP, DOWN, LEFT, and RIGHT. Other keyboard functions are used to accelerate/decelerate the animation (+/-), zoom in and out of the animation (z/Z), enable and disable the particle system (p), enable and disable the blending function (b).

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, DLL files, glu32.dll, glut32.dll, opengl32.dll.

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.

- **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.

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

This 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 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 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.

- **glLoadName (ID)**

To replace the top of the integer-ID name stack for picking operations.

- **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 integer 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 matrix on top of the stack and to make the second matrix on the stack become the current matrix.

- **glutKeyboardFunc (keyboardfcn)**

Specify a keyboard callback function that is to be invoked when a standard key is pressed.

- **glutSpecialFunc (NonASCIIKeyboardPress)**

This function sets the special keyboard callback for the current window.

- **glShadeModel (GLenum mode)**

This function selects the flat or smooth shading.

- **glDepthFunc (GLenum func)**

This function specifies the value used for depth buffer comparisons.

- **glHint (GLenum target, GLenum mode**)

This function specifies the implementation-specific hints.

- **glCullFace (GLenum mode)**

This function specifies whether front- or back-facing facets can be culled.

- **glPixelStorei (GLenum pname, GLint param)**

This function sets the pixel storage modes.

- **glutIgnoreKeyRepeat(false)**

This function determines if auto repeat keystrokes are reported to the current window.

- **glLightModelfv (GLenum pname, const GLfixed * params)**

This function sets the values of individual light source parameters.

- **gluPerspective (GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)**

This function sets up a perspective projection matrix.

- **gluLookAt (GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ, GLdouble centerX, GLdouble centerY, GLdouble centerZ, GLdouble upX, GLdouble upY, GLdouble upZ)**

This function defines a viewing transformation.

- **glDepthMask (GLboolean flag)**

This function enables or disables writing into the depth buffer.

- **glBlendFunc (GLenum sfactor, GLenum dfactor)**

This function specifies the pixel arithmetic.

- **glutSwapBuffers ()**

This function swaps the buffers of the current window if double buffered.

- **glFlush ()**

This function force execution of GL commands in finite time.

- **gluQuadricNormals (GLUquadric* quad, Glenum normal)**

This function specifies what kind of normals are desired for quadrics.

- **gluQuadricTexture (GLUquadric* quad, Glboolean texture)**

This function specifies if texturing is desired for quadrics.

- **glTexEnvf (Glenum target, Glenum pname, Glenum param)**

This function set texture environment parameters.

- **glRotatef (GLfloat angle, GLfloat x, GLfloat y, GLfloat z)**

This function multiplies the current matrix by a rotation matrix.

- **glScalef (GLfloat x, GLfloat y, GLfloat z)**

This function multiplies the current matrix by a general scaling matrix.

- **glTranslatef (GLfloat x, GLfloat y, GLfloat z)**

This function multiplies the current matrix by a translation matrix.

- **void glutTimerFunc (unsigned int msecs, void (*func) (int value), values**:

glutTimerFunc registers a timer call-back to be triggered in a specified number of milliseconds.

- **glBindTexture (GLenum texture, Glint texture)**

This function binds a named texture to a texturing target.

- **gluSphere (GLUquadric* quad, GLdouble radius, GLint slices, GLint stacks)**

This function draws a sphere.

- **gluDeleteQuadric (GLUquadric* quad)**

This function destroys a quadrics object. quad specifies the quadrics object to be destroyed.

- **glTexParameteri (GLenum target, GLenum pname, GLint param)**

This function sets texture parameters.

- **glTexImage2D (GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const void * data)**

This function specifies a two-dimensional texture image.

- **glCopyPixels (x, y, width, height, GL_COLOR)**

glCopyPixels copies a screen-aligned rectangle of pixels from the specified frame buffer location to a region relative to the current raster position.

- **glDrawPixels (GLsizei width, GLsizei height, GLenum format, GLenum type, const void * data)**

This function writes a block of pixels to the frame buffer.

- **glReadPixels (GLint x, GLint y, width, GLsizei height, GLenum format, GLenum type, void * data)**

This function reads a block of pixels from the frame buffer.

User-Defined Functions:

- ### KeyBoardPress ():

void KeyboardPress (unsigned char pressedKey, int mouseXPosition, int mouseYPosition)

Function to react to ASCII keyboard keys pressed by the user.

void NonASCIIKeyboardPress (int pressedKey, int mouseXPosition, int mouseYPosition)

Function to react to non-ASCII keyboard keys pressed by the user.

- ### TimerFunction ():

void TimerFunction (int value)

The EarthDayIncrement represents the fraction of an Earth day being added to the scene in one screen update using TimerFunction ().

- ### Display ():

void Display ()

Principal display routine: sets up material, lighting, and camera properties, clears the frame buffer, and draws all texture-mapped objects within the window.

- ### MakeAllImages ():

void MakeAllImages ()

This function creates the textures associated with all texture-mapped objects being displayed.

- ### MakeImage ():

void MakeImage (const char bitmapFilename [], GLuint& textureName, bool hasAlpha)

This function converts the bitmap with the parameterized name into an OpenGL texture.

- ## SetLights ():

void SetLights ()

This function sets the two lights to illuminate the scene.

- ## UpdateLight ():

void UpdateLight ()

This function enables the scene's lighting.

- ## drawEarthAndMoon ():

void drawEarthAndMoon ()

This function draws the texture-mapped earth and moon.

- ## drawSun ():

void drawSun ()

Function to draw and texture map the sun at the origin.

- ## drawSaturnRing ():

void drawSaturnRing ()

This function draws the Saturn Rings.

- ## drawAllPlanets ():

void drawAllPlanets ()

This function makes calls to the generic planet drawing function. It is not included in the display function to enhance readability.

- ## drawGenericPlanet ():

void drawGenericPlanet (GLfloat inclination, GLfloat orbitDuration,
GLfloat orbitRadius, GLfloat rotationDuration, GLuint texturename, GLfloat radius)

Given parameters about the planets dimension, orbit, radius etc, this function will draw a texture mapped plant. It is used to draw everything except the sun, earth/moon and Saturn rings, as they are special cases of this function.

- **drawParticle ():**

void drawParticle (Particle currParticle)

This function draws one individual particle, given a particle struct from the particle system object.

- **drawAllParticles ():**

void drawAllParticles ()

This function cycles through each particle in the particle system and passes it to the draw function.

- **IntroductionTimerFunction ():**

void IntroductionTimerFunction(int)

This timer function is to set display function, reshape function and enable depth test after introduction screen.

- **displayText ():**

void displayText (float, float, int, int, int, const char*)

This function displays the text in the introduction screen.

- **myinit ():**

void myinit ()

This function executes initialization procedure.

- **Display_Intro ():**

void Display_Intro ()

This function is to display the contents of introduction screen.

- **setTexture ():**

void setTexture (GLuint textureName)

This function binds a name texture to texturing target and sets texture parameters.

- **readBMPFile ():**

void readBMPFile (string fname, bool hasAlpha)

This function reads an mRGB image from an uncompressed BMP file into memory and returns 0 on failure, 1 on success.

- **getLong ():**

void getLong ():

This helper function is used to convert little-endian to bid endian as BMP format uses little-endian integer types.

- **getShort ():**

void getShort ()

This helper function is used to convert little-endian to bid endian as BMP format uses little-endian integer types.

- **freeIt ():**

void freeIt ()

This function gives back memory for this pixmap.

- **read ():**

int read (int x, int y, int wid, int ht)

This function reads a rectangle of pixels into this pixmap.

int read (IntRect r)

This function reads a rectangle of pixels into this pixmap.

- **setPixel ();**

void setPixel (int x, int y, mRGB color)

This function sets the pixel at a specified (x, y) co-ordinate to a particular color.

- **getPixel ():**

mRGB getPixel (int x, int y)

This function returns the color of pixel present at location (x, y).

- **draw ():**

void draw ()

This function draws the pixmap at current raster position.

- **copy ():**

void copy (IntPoint from, IntPoint to, int x, int y, int width, int height)

This function copies a region of the display back onto the display.

- **Particle System**

Particle getNextParticle ()

This function returns the next particle in the array.

- **updateAll ():**

void updateAll ()

if the particle's lifetime is over (0), replace it with another (create new particle), otherwise update all the appropriate values (Modify all particles by their deltas).

- **getNumberOfParticles ():**

int getNumberOfParticles ()

This function returns number of particles used i.e., 3000.

- **ParticleSystem ();**

Constructor used to generate all particles.

- **generateRandomNumber ():**

float generateRandomNumber (float lower, float upper)

This function generates random number between lower and upper value.

- **generateNewParticle ():**

Particle generateNewParticle ()

This function gets a new Particle struct with new randomized values. It creates new particle by initialising azimuthal rotation, zenithal rotation, surface translation factor, delta values and lifetime.

- **time ():**

time(&randomNumberSeed)

This function calculates the number of seconds from jan 1st 1970 till the current time.

- **srand ():**

srand (randomNumberSeed)

This function sets the starting point for producing a series of pseudo-random integers.

- **ResizeWindow():**

void ResizeWindow(GLsizei w, GLsizei h)

Window-reshaping callback, adjusting the viewport to be as large as possible within the window, without changing its aspect ratio.

3.2 PSEUDOCODE/FLOWCHART/FLOW DIAGRAMS

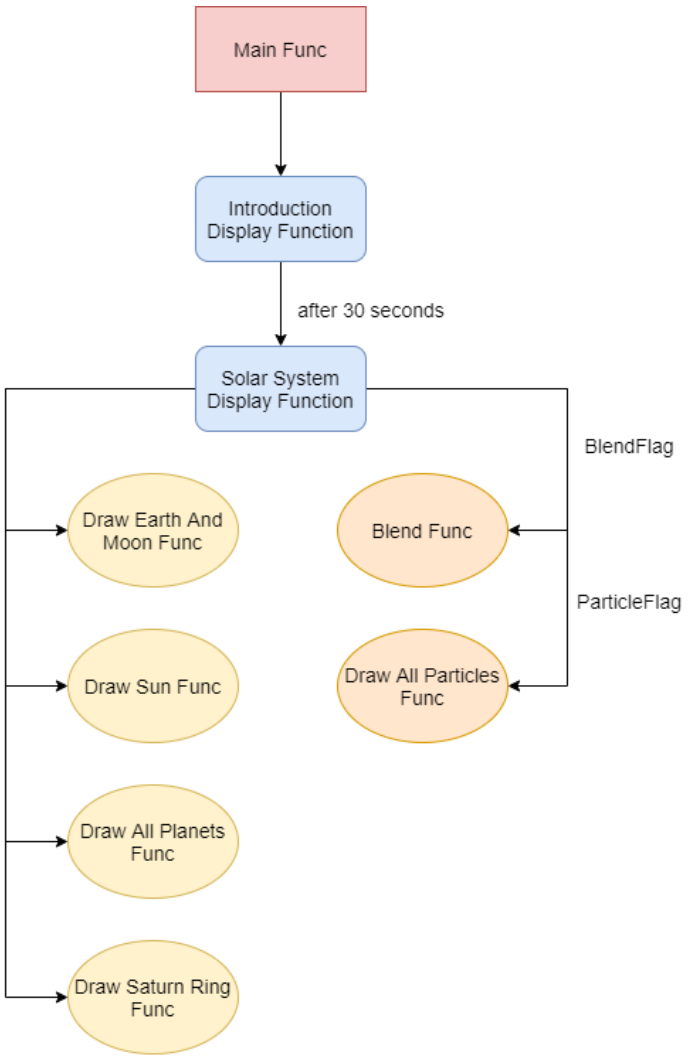

Fig: 3.1 Solar System Flowchart

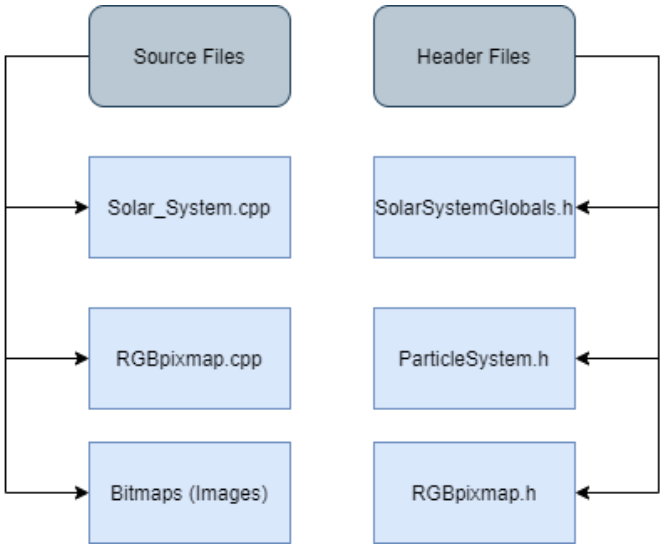

Fig: 3.2 File Structure

CHAPTER 4

IMPLEMENTATION

4.1 Solar_System.cpp

4.1.1 Main Function:

The main function sets up the data and the environment to display the textured objects.

```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);

   // Set up the display window.
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_STENCIL | GLUT_DEPTH);
   glutInitWindowPosition(INIT_WINDOW_POSITION[0], INIT_WINDOW_POSITION[1]);
   glutInitWindowSize(currWindowSize[0], currWindowSize[1]);
   glutCreateWindow("Solar System");
   glutDisplayFunc(Display_Intro);
   //after 30 seconds solar system is displayed
   glutTimerFunc(30000, IntroductionTimerFunction, 1);
   glutKeyboardFunc(KeyboardPress);
   glutSpecialFunc(NonASCIIKeyboardPress);
   glutTimerFunc(20, TimerFunction, 1);
   glViewport(0, 0, currWindowSize[0], currWindowSize[1]);

   // Set up standard lighting, shading, and depth testing.
   glShadeModel(GL_SMOOTH);
   glDepthFunc(GL_LEQUAL);
   glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);     glEnable(GL_NORMALIZE);
   glCullFace(GL_BACK);
   glEnable(GL_CULL_FACE);
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
   SetLights();

   // Set up all texture maps and texture-mapped objects.
   MakeAllImages();
   //set up sound effect
   bool played = PlaySound("SolarSystem_Sound.wav", NULL, SND_LOOP | SND_ASYNC);
   glutMainLoop();
   return 0;
}
```

4.1.2 Introduction Timer Function:

This timer function is to set display function, reshape function, and enable depth test after introduction screen.

```
void IntroductionTimerFunction(int value)
{
   if (value == 1)
   {
      glutDisplayFunc(Display);
      glutReshapeFunc(ResizeWindow);
      glEnable(GL_DEPTH_TEST);
   }
}
```

4.1.3 Display Function:

Principal display routine: sets up material, lighting, and camera properties, clears the frame buffer, and draws all texture-mapped objects within the window.

```
void Display()
{
   // Initialize lighting.
   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, LIGHT_MODEL_AMBIENT);
   glEnable(GL_LIGHTING);

   // Set up the properties of the viewing camera.
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60.0, ASPECT_RATIO, 0.2, 100.0);
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();

   // Position and orient viewer.
   gluLookAt(LOOK_AT_POSITION[0] + ViewerDistance * sin(viewerZenith) *
sin(viewerAzimuth),
      LOOK_AT_POSITION[1] + ViewerDistance * cos(viewerZenith),
      LOOK_AT_POSITION[2] + ViewerDistance * sin(viewerZenith) * cos(viewerAzimuth),
      LOOK_AT_POSITION[0], LOOK_AT_POSITION[1], LOOK_AT_POSITION[2],
      0.0, 1.0, 0.020);

   // Render scene.
   UpdateLight();
   drawEarthAndMoon();
   drawSun();
   drawAllPlanets();
   drawSaturnRing();

   //enable blending
   if (blendFlag) {
      glEnable(GL_BLEND);
      glDepthMask(GL_FALSE);
      glBlendFunc(GL_SRC_COLOR, GL_ONE);
   }
   if (particleFlag)
      drawAllParticles();

   glDepthMask(GL_TRUE);
   glDisable(GL_BLEND);
   glDisable(GL_LIGHTING);
   glutSwapBuffers();
   glFlush();
}
```

4.1.4 Draw All Planets Function:

This function makes call to the generic planet drawing function. It is not included in the display function to enhance readability.

```
void drawAllPlanets() {
   drawGenericPlanet(MERCURY_INCLINATION, MERCURY_ORBIT_DUR, MERCURY_ORBIT_RADIUS,
      MERCURY_ROTATION_DUR, MercuryTextureName, MERCURY_RADIUS);
   drawGenericPlanet(VENUS_INCLINATION, VENUS_ORBIT_DUR, VENUS_ORBIT_RADIUS,
      VENUS_ROTATION_DUR, VenusTextureName, VENUS_RADIUS);
   drawGenericPlanet(MARS_INCLINATION, MARS_ORBIT_DUR, MARS_ORBIT_RADIUS,
      MARS_ROTATION_DUR, MarsTextureName, MARS_RADIUS);
   drawGenericPlanet(JUPITER_INCLINATION, JUPITER_ORBIT_DUR, JUPITER_ORBIT_RADIUS,
      JUPITER_ROTATION_DUR, JupiterTextureName, JUPITER_RADIUS);
   drawGenericPlanet(SATURN_INCLINATION, SATURN_ORBIT_DUR, SATURN_ORBIT_RADIUS,
      SATURN_ROTATION_DUR, SaturnTextureName, SATURN_RADIUS);
   drawGenericPlanet(URANUS_INCLINATION, URANUS_ORBIT_DUR, URANUS_ORBIT_RADIUS,
      URANUS_ROTATION_DUR, UranusTextureName, URANUS_RADIUS);
   drawGenericPlanet(NEPTUNE_INCLINATION, NEPTUNE_ORBIT_DUR, NEPTUNE_ORBIT_RADIUS,
      NEPTUNE_ROTATION_DUR, NeptuneTextureName, NEPTUNE_RADIUS);

}
```

4.1.5 Draw Earth and Moon Function:

Draws the texture-mapped earth and moon.

```
void drawEarthAndMoon()
{
   GLfloat MoonRevolution = EarthDaysTranspired / LUNAR_CYCLE;
   GLUquadricObj* quadro = gluNewQuadric();
   gluQuadricNormals(quadro, GLU_SMOOTH);
   gluQuadricTexture(quadro, GL_TRUE);
   glEnable(GL_TEXTURE_2D);
   glPushMatrix();
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
   glPushMatrix();
   glRotatef(EARTH_INCLINATION, 0.0, 0.0, 1.0);
   glRotatef(360.0 * (EarthDaysTranspired / EARTH_ORBIT_DUR), 0.0, 1.0, 0.0);
   glTranslatef(EARTH_ORBIT_RADIUS, 0.0, 0.0);
   glRotatef(360.0 * CurrentEarthRotation, 0.0, 1.0, 0.0);
   glRotatef(-90.0, 1.0, 0.0, 0.0);
   glBindTexture(GL_TEXTURE_2D, EarthTextureName);
   gluSphere(quadro, EARTH_RADIUS, 48, 48);
   glPopMatrix();
   glRotatef(EARTH_INCLINATION, 0.0, 0.0, 1.0);
   glRotatef(360.0 * (EarthDaysTranspired / EARTH_ORBIT_DUR), 0.0, 1.0, 0.0);
   glTranslatef(EARTH_ORBIT_RADIUS, 0.0, 0.0);
   glRotatef(360.0 * MoonRevolution, 0.0, 1.0, 0.0);
   glTranslatef(MOON_ORBIT_RADIUS, 0.0, 0.0);
   glBindTexture(GL_TEXTURE_2D, MoonTextureName);
   gluSphere(quadro, MOON_RADIUS, 48, 48);
   glPopMatrix();
   glDisable(GL_TEXTURE_2D);
   gluDeleteQuadric(quadro);
}
```

4.1.6 Draw Sun Function:

Function to draw and texture map the sun at the origin.

```
void drawSun()
{
   GLUquadricObj* quadro = gluNewQuadric();
   gluQuadricNormals(quadro, GLU_SMOOTH);
   gluQuadricTexture(quadro, GL_TRUE);
   glEnable(GL_TEXTURE_2D);
   glPushMatrix();
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
   glPushMatrix();
   glRotatef(-90.0, 1.0, 0.0, 0.0);
   glBindTexture(GL_TEXTURE_2D, SunTextureName);
   gluSphere(quadro, SUN_RADIUS, 48, 48);
   glPopMatrix();
   glPopMatrix();
   glDisable(GL_TEXTURE_2D);
   gluDeleteQuadric(quadro);
}
```

4.1.7 Draw Saturn Ring Function:

```
void drawSaturnRing()
{
   GLUquadricObj* quadro = gluNewQuadric();
   gluQuadricNormals(quadro, GLU_SMOOTH);
   gluQuadricTexture(quadro, GL_TRUE);
   glEnable(GL_TEXTURE_2D);
   glPushMatrix();
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
   glPushMatrix();
   glRotatef(SATURN_INCLINATION, 0.0, 0.0, 1.0);
   glRotatef(360.0 * (EarthDaysTranspired / SATURN_ORBIT_DUR), 0.0, 1.0, 0.0);
   glTranslatef(SATURN_ORBIT_RADIUS, 0.0, 0.0);
   glRotatef(-90.0, 1.0, 0.0, 0.0);
   glBindTexture(GL_TEXTURE_2D, RingTextureName);
   glScalef(1, 1, .02);
   gluSphere(quadro, SATURN_RADIUS * 2, 48, 48);
   glPopMatrix();
   glPopMatrix();
   glDisable(GL_TEXTURE_2D);
   gluDeleteQuadric(quadro);
}
```

4.1.8 Draw Generic Planet Function:

Given parameters about the planets dimension, orbit, radius etc., this function will draw a texture mapped plant. It is used to draw everything except the sun, earth/moon and Saturn's rings, as they are special cases of this function.

```
void drawGenericPlanet(GLfloat inclination, GLfloat orbitDuration,
   GLfloat orbitRadius, GLfloat rotationDuration, GLuint texturename, GLfloat radius)
{
   GLUquadricObj* quadro = gluNewQuadric();
   gluQuadricNormals(quadro, GLU_SMOOTH);
```

```
   gluQuadricTexture(quadro, GL_TRUE);
   glEnable(GL_TEXTURE_2D);
   glPushMatrix();
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
   glPushMatrix();
   glRotatef(inclination, 0.0, 0.0, 1.0);
   glRotatef(360.0 * (EarthDaysTranspired / orbitDuration), 0.0, 1.0, 0.0);
   glTranslatef(orbitRadius, 0.0, 0.0);
   glRotatef(360.0 * (CurrentEarthRotation / rotationDuration), 0.0, 1.0, 0.0);
   glRotatef(-90.0, 1.0, 0.0, 0.0);
   glBindTexture(GL_TEXTURE_2D, texturename);
   gluSphere(quadro, radius, 48, 48);
   glPopMatrix();
   glPopMatrix();
   glDisable(GL_TEXTURE_2D);
   gluDeleteQuadric(quadro);
}
```

4.1.9 Draw All Particles Function:

Cycles through each particle in the particle system and passes it to the draw function.

```
void drawAllParticles() {
   particles.updateAll();
   for (int i = 0; i < particles.getNumberOfParticles(); i++)
      drawParticle(particles.getNextParticle());

}
```

4.1.10 Draw Particle Function:

Draws one individual particle, given a particle struct from the particle system object.

```
void drawParticle(Particle currParticle)
{
   glEnable(GL_TEXTURE_2D);
   glPushMatrix();
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
   glRotatef(currParticle.azimuthRotation, 0, 1, 0);
   glRotatef(currParticle.zenithRotation, 0, 0, 1);
   glTranslatef(SUN_RADIUS + currParticle.surfaceTranslationFactor, 0, 0);
   glRotatef(90, 0, 1, 0);
   glScalef(.5, .5, 1.0);
   glBindTexture(GL_TEXTURE_2D, ParticleTextureName);


   glBegin(GL_TRIANGLE_STRIP);
   glTexCoord2d(1, 1);
   glVertex3f(0.5f, 0.5f, 0.0f); // Top Right
   glTexCoord2d(0, 1);
   glVertex3f(-0.5f, 0.5f, 0.0f); // Top Left
   glTexCoord2d(1, 0);
   glVertex3f(0.5f, -0.5f, 0.0f); // Bottom Right
   glTexCoord2d(0, 0);
   glVertex3f(-0.5f, -0.5f, 0.0f); // Bottom Left
   glEnd();
```

```
   glBegin(GL_TRIANGLE_STRIP);
   glTexCoord2d(1, 1);
   glVertex3f(-0.5f, 0.5f, 0.0f); // Top Right
   glTexCoord2d(0, 1);
   glVertex3f(0.5f, 0.5f, 0.0f); // Top Left
   glTexCoord2d(1, 0);
   glVertex3f(-0.5f, -0.5f, 0.0f); // Bottom Right
   glTexCoord2d(0, 0);
   glVertex3f(0.5f, -0.5f, 0.0f); // Bottom Left
   glEnd();
   glPopMatrix();
   glDisable(GL_TEXTURE_2D);
}
```

4.1.11 Keyboard Press Function:

Function to react to ASCII keyboard keys pressed by the user.

```
void KeyboardPress(unsigned char pressedKey, int mouseXPosition, int mouseYPosition)
{
   char pressedChar = char(pressedKey);
   switch (pressedKey)
   {
      case '+': {                              //+ key is used to accelerate
         EarthDayIncrement *= 2.0;
         if (EarthDayIncrement > 10.0)
            EarthDayIncrement = 10.0;
         break;
      }
      case '-': {                             //- keys is used to decelerate
         EarthDayIncrement *= 0.5;
         if (EarthDayIncrement < 0.01)
            EarthDayIncrement = 0.01;
         break;
      }
      case 'z': {                            //z key is used to zoom in
         ViewerDistance -= VIEWER_DISTANCE_INCREMENT;
         if (ViewerDistance < MINIMUM_VIEWER_DISTANCE)
            ViewerDistance = MINIMUM_VIEWER_DISTANCE;
         break;
      }
      case 'Z': {                           //Z key is used to zoom  out
         ViewerDistance += VIEWER_DISTANCE_INCREMENT;
         if (ViewerDistance > MAXIMUM_VIEWER_DISTANCE)
            ViewerDistance = MAXIMUM_VIEWER_DISTANCE;
         break;
      }
            //enable and disable the particle system
      case 'p': {
         particleFlag = !particleFlag;
         break;
      }
            //enable and disable the blending function
      case 'b': {
         blendFlag = !blendFlag;
         break;
      }
   }
}
```

4.1.12 Non-ASCII Keyboard Press Function:

Function to react to non-ASCII keyboard keys pressed by the user.

```
void NonASCIIKeyboardPress(int pressedKey, int mouseXPosition, int mouseYPosition)
{
   glutIgnoreKeyRepeat(false);
   switch (pressedKey)
   {
      case GLUT_KEY_RIGHT: {
         viewerAzimuth += VIEWER_ANGLE_INCREMENT;
         if (viewerAzimuth > 2 * PI)
            viewerAzimuth -= 2 * PI;
         break;
      }
      case GLUT_KEY_LEFT: {
         viewerAzimuth -= VIEWER_ANGLE_INCREMENT;
         if (viewerAzimuth < 0.0)
            viewerAzimuth += 2 * PI;
         break;
      }
      case GLUT_KEY_UP: {
         viewerZenith -= VIEWER_ANGLE_INCREMENT;
         if (viewerZenith < VIEWER_ANGLE_INCREMENT)
            viewerZenith = VIEWER_ANGLE_INCREMENT;
         break;
      }
      case GLUT_KEY_DOWN: {
         viewerZenith += VIEWER_ANGLE_INCREMENT;
         if (viewerZenith > PI - VIEWER_ANGLE_INCREMENT)
            viewerZenith = PI - VIEWER_ANGLE_INCREMENT;
         break;
      }
   }
}
```

4.1.13 Timer Function:

The EarthDayIncrement represents the fraction of an Earth Day being added to the scene in one screen update.

```
void TimerFunction(int value)
{
   CurrentEarthRotation += EarthDayIncrement;
   EarthDaysTranspired += EarthDayIncrement;
   if (EarthDaysTranspired == EARTH_ORBIT_DUR)
      EarthDaysTranspired = 0;
   glutPostRedisplay();
   glutTimerFunc(20, TimerFunction, 1);
}
```

4.1.14 Resize Window Function:

Window-reshaping callback, adjusting the viewport to be as large as possible within the window, without changing its aspect ratio.

```
void ResizeWindow(GLsizei w, GLsizei h)
{
   currWindowSize[0] = w;
   currWindowSize[1] = h;
   if (ASPECT_RATIO > w / h)
   {
      currViewportSize[0] = w;
      currViewportSize[1] = w / ASPECT_RATIO;
   }
   else
   {
      currViewportSize[1] = h;
      currViewportSize[0] = h * ASPECT_RATIO;
   }

   glViewport(0.5 * (w - currViewportSize[0]), 0.5 * (h - currViewportSize[1]),
currViewportSize[0], currViewportSize[1]);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 0.1, 100.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

4.1.15 Make All Images Function:

Create the textures associated with all texture-mapped objects being displayed.

```
void MakeAllImages()
{
   MakeImage(EARTH_BMP_FILENAME, EarthTextureName, false);
   MakeImage(MOON_BMP_FILENAME, MoonTextureName, false);
   MakeImage(SUN_BMP_FILENAME, SunTextureName, false);
   MakeImage(MERCURY_BMP_FILENAME, MercuryTextureName, false);
   MakeImage(VENUS_BMP_FILENAME, VenusTextureName, false);
   MakeImage(PARTICLE_BMP_FILENAME, ParticleTextureName, false);
   MakeImage(MARS_BMP_FILENAME, MarsTextureName, false);
   MakeImage(JUPITER_BMP_FILENAME, JupiterTextureName, false);
   MakeImage(SATURN_BMP_FILENAME, SaturnTextureName, false);
   MakeImage(URANUS_BMP_FILENAME, UranusTextureName, false);
   MakeImage(NEPTUNE_BMP_FILENAME, NeptuneTextureName, false);
   MakeImage(RING_BMP_FILENAME, RingTextureName, false);
   return;
}
```

4.1.16 Make Image Function:

Convert the bitmap with the parameterized name into an OpenGL texture.

```
void MakeImage(const char bitmapFilename[], GLuint& textureName, bool hasAlpha)
{
   RGBpixmap pix;
   pix.readBMPFile(bitmapFilename, hasAlpha);
   pix.setTexture(textureName);
   return;
}
```

4.1.17 Set Lights Function:

Set the two lights to illuminate the scene.

```
void SetLights()
{
   glLightfv(GL_LIGHT0, GL_AMBIENT, LIGHT_AMBIENT);
   glLightfv(GL_LIGHT0, GL_DIFFUSE, LIGHT_DIFFUSE);
   glLightfv(GL_LIGHT0, GL_SPECULAR, LIGHT_SPECULAR);
   glLightfv(GL_LIGHT0, GL_POSITION, LIGHT_0_POSITION);

   glLightfv(GL_LIGHT1, GL_AMBIENT, LIGHT_AMBIENT);
   glLightfv(GL_LIGHT1, GL_DIFFUSE, LIGHT_DIFFUSE);
   glLightfv(GL_LIGHT1, GL_SPECULAR, LIGHT_SPECULAR);
   glLightfv(GL_LIGHT1, GL_POSITION, LIGHT_1_POSITION);

   glLightfv(GL_LIGHT2, GL_AMBIENT, LIGHT_AMBIENT);
   glLightfv(GL_LIGHT2, GL_DIFFUSE, LIGHT_DIFFUSE);
   glLightfv(GL_LIGHT2, GL_SPECULAR, LIGHT_SPECULAR);
   glLightfv(GL_LIGHT2, GL_POSITION, LIGHT_2_POSITION);

   glLightfv(GL_LIGHT3, GL_AMBIENT, LIGHT_AMBIENT);
   glLightfv(GL_LIGHT3, GL_DIFFUSE, LIGHT_DIFFUSE);
   glLightfv(GL_LIGHT3, GL_SPECULAR, LIGHT_SPECULAR);
   glLightfv(GL_LIGHT3, GL_POSITION, LIGHT_3_POSITION);
}
```

4.1.18 Update Light Function:

Enable the scene's lighting.

```
void UpdateLight()
{
   glPushMatrix();
   glLightfv(GL_LIGHT0, GL_POSITION, LIGHT_0_POSITION);
   glLightfv(GL_LIGHT1, GL_POSITION, LIGHT_1_POSITION);
   glLightfv(GL_LIGHT2, GL_POSITION, LIGHT_2_POSITION);
   glLightfv(GL_LIGHT3, GL_POSITION, LIGHT_3_POSITION);
   glPopMatrix();
   glEnable(GL_LIGHT0);
   glEnable(GL_LIGHT1);
}
```

4.2 RGBpixmap.cpp

RGBpixmap.cpp - routines to read a BMP file.

4.2.1 Read BMP File Function:

```
int RGBpixmap::readBMPFile(string fname, bool hasAlpha)
{
   // Read into memory an mRGB image from an uncompressed BMP file.
   // Return 0 on failure, 1 on success
   inf.open(fname.c_str(), ios::in | ios::binary);
   if (!inf) { cout << " can't open file: " << fname << endl; return 0; }
   int k, row, col, numPadBytes, nBytesInRow;

   // read the file header information
   char ch1, ch2;
   inf.get(ch1);
   inf.get(ch2);              //type: always 'BMP'
   ulong fileSize = getLong();
   ushort reserved1 = getShort();   // always 0
   ushort reserved2 = getShort();   // always 0
   ulong offBits = getLong(); // offset to image - unreliable
   ulong headerSize = getLong(); // always 40
   ulong numCols = getLong(); // number of columns in image
   ulong numRows = getLong(); // number of rows in image
   ushort planes = getShort();// always 1
       ushort bitsPerPixel = getShort(); //8 or 24; allow 24 here -> 24-bit(RGB=8*3)
       image(a true color image) => 16M colors
   ulong compression = getLong();   // must be 0 for uncompressed
   ulong imageSize = getLong();  // total bytes in image
   ulong xPels = getLong(); // always 0
   ulong yPels = getLong(); // always 0
   ulong numLUTentries = getLong(); // 256 for 8 bit, otherwise 0
   ulong impColors = getLong();  // always 0

   if (bitsPerPixel != 24)
   {
      // error - must be a 24 bit uncompressed image
      cout << "not a 24 bit/pixelimage, or is compressed!\n";
      inf.close();
      return 0;
   }


   nBytesInRow = ((3 * numCols + 3) / 4) * 4;
   numPadBytes = nBytesInRow - 3 * numCols; // number of padding bytes required
   nRows = numRows;
   nCols = numCols;
   pixel = new mRGB[nRows * nCols];
   if (!pixel)
      return 0;

   long count = 0;
   char dum;
   for (row = 0; row < nRows; row++) // read pixel values
   {
      for (col = 0; col < nCols; col++)
      {
         char r, g, b;
         inf.get(b);
```

```
         inf.get(g);
         inf.get(r);
         pixel[count].r = r;
         pixel[count].g = g;
         pixel[count].b = b;
         if ((hasAlpha) && (r == -1) && (g == -1) && (b == -1))
   pixel[count++].a = 0;
         else
            pixel[count++].a = 255;
      }
      for (k = 0; k < numPadBytes; k++)
         inf >> dum;
   }
   inf.close();
   return 1;
}
```

4.2.2 Set Texture Function:

This function binds a name texture to texturing target and sets texture parameters.

```
void RGBpixmap::setTexture(GLuint textureName)
{
   glBindTexture(GL_TEXTURE_2D, textureName);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, nCols, nRows, 0, GL_RGBA, GL_UNSIGNED_BYTE,
pixel);

}
```

4.3 ParticleSystem.h

```
struct Particle {
   float azimuthRotation, zenithRotation, surfaceTranslationFactor;
   float deltaAz, deltaZe, deltaSurface;
   //Lifetime In "Refreshes"
   int lifetime;
};

class ParticleSystem {
public:

   Particle getNextParticle();
   void updateAll();
   int getNumberOfParticles();
   ParticleSystem();

private:
   int currentParticle;

   //Array of all particle structs
   Particle particles[NUMBER_OF_PARTICLES];
   float generateRandomNumber(float lower, float upper);
   Particle generateNewParticle();
};

//constructor-creates 3000 particles and stored in array of particles
```

```
ParticleSystem::ParticleSystem() {
   currentParticle = 0;
   for (int i = 0; i < NUMBER_OF_PARTICLES; i++) {
      particles[i] = generateNewParticle();
   }
}

//Get a new Particle struct with new randomized values.
Particle ParticleSystem::generateNewParticle() {
   Particle newParticle;
   newParticle.azimuthRotation = generateRandomNumber(MIN_ROTATION, MAX_ROTATION);
   newParticle.zenithRotation = generateRandomNumber(MIN_ROTATION, MAX_ROTATION);
   newParticle.surfaceTranslationFactor = generateRandomNumber(MIN_TRANS, MAX_TRANS);
   newParticle.deltaAz = generateRandomNumber(MIN_DELTA_ROTATION, MAX_DELTA_ROTATION);
   newParticle.deltaZe = generateRandomNumber(MIN_DELTA_ROTATION, MAX_DELTA_ROTATION);
   newParticle.deltaSurface = generateRandomNumber(MIN_DELTA_SURFACE,
MAX_DELTA_SURFACE);
   newParticle.lifetime = (int)generateRandomNumber(MIN_LIFETIME, MAX_LIFETIME);
   return newParticle;
}

//generating random number btw lower and upper values
float ParticleSystem::generateRandomNumber(float lower, float upper) {
   static bool firstTime = true;
   time_t randomNumberSeed;
   if (firstTime) {
      time(&randomNumberSeed);
      srand(randomNumberSeed);
      firstTime = false;            }
   return (lower + ((upper - lower) * (float(rand()) / RAND_MAX)));
   //(a+((b-a)*(rand()/rand_max))
}

//returns the next particle in the array
Particle ParticleSystem::getNextParticle() {
   currentParticle = (currentParticle == NUMBER_OF_PARTICLES) ? 0 : currentParticle;
   return particles[currentParticle++];
}

//returns 3000 (number of max particles)
int ParticleSystem::getNumberOfParticles() {
   return NUMBER_OF_PARTICLES;
}

//Modify all particles by their deltas, destroy old ones and create new ones, if the
lifetime is 0
void ParticleSystem::updateAll() {
   for (int i = 0; i < NUMBER_OF_PARTICLES; i++) {
      if (particles[i].lifetime == 0) {
         particles[i] = generateNewParticle();
      }
      else {
         particles[i].azimuthRotation += particles[i].deltaAz;
         particles[i].zenithRotation += particles[i].deltaZe;
         particles[i].surfaceTranslationFactor += particles[i].deltaSurface;
         particles[i].lifetime -= DELTA_LIFETIME;
      }
   }
}
```

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**	• Key z to zoom in of the animation, • Key Z to zoom out of the animation, • Key p to enable and disable the particle system, • Key b to enable and disable the blending function, • Key + to accelerate the animation, • Key – to decelerate the animation, • KEY_UP/KEY_DOWN/KEY_RIGHT/ KEY_LEFT to alter spherical coordinates of the viewer's position. (Top View, Front View)	Results obtained as expected
2	**Display Function**	Sun and eight planets revolving around the sun, particle system	Results obtained as expected
3	**Animation Effect**	Textures, lighting, and transformation operations like translation, rotation, scaling on objects.	Results obtained as expected

4	**Sound Effect**	Plays .wav format audio in the background	Results obtained as expected
5	**Texture Mapping**	Applying high frequency detail, surface texture, or color information on a computer-generated graphic or 3D sphere.	Results obtained as expected

Table 5.1: Testing

5.2 SCREENSHOTS/SNAPSHOTS

5.2.1 Introduction:

The below figure represents the introductory page of "SOLAR SYSTEM" mini project. glutBitmapCharacter( ) is used to render the text on the screen.

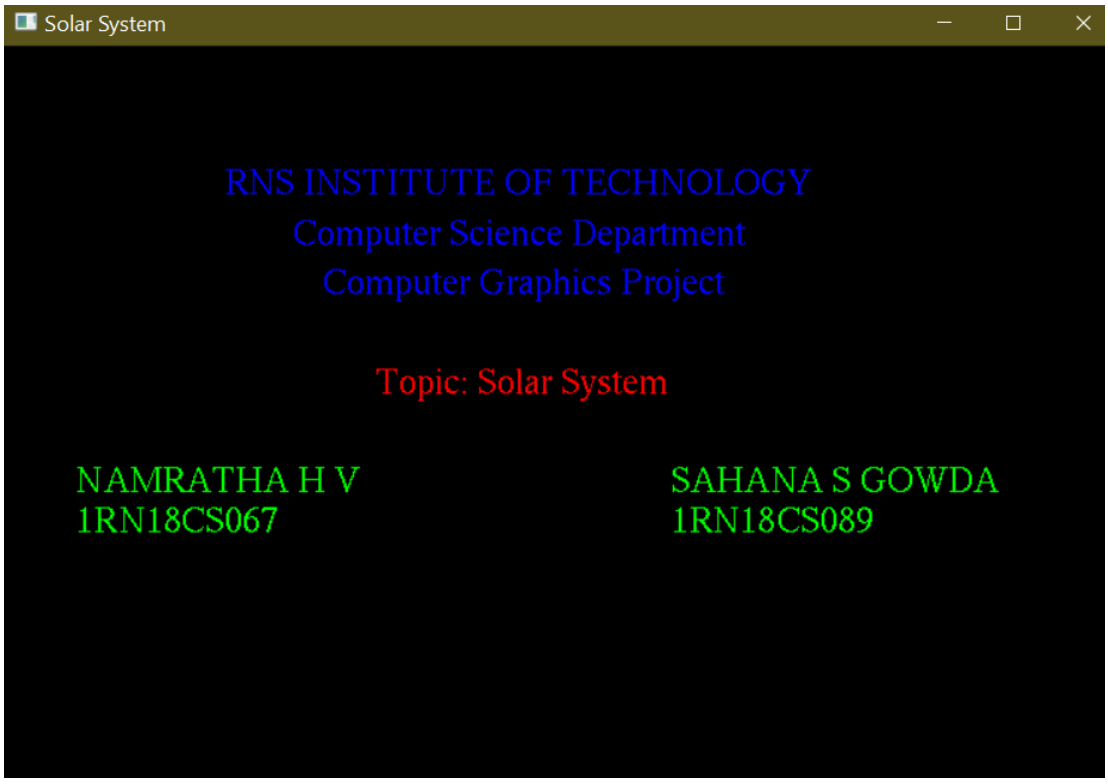

Fig: 5.1 Introduction

5.2.2 Final view of Solar System:

In this snapshot, sun is placed at the centre and its eight planets are placed in the sun's orbit. These eight planets are shown to be revolving around the sun.

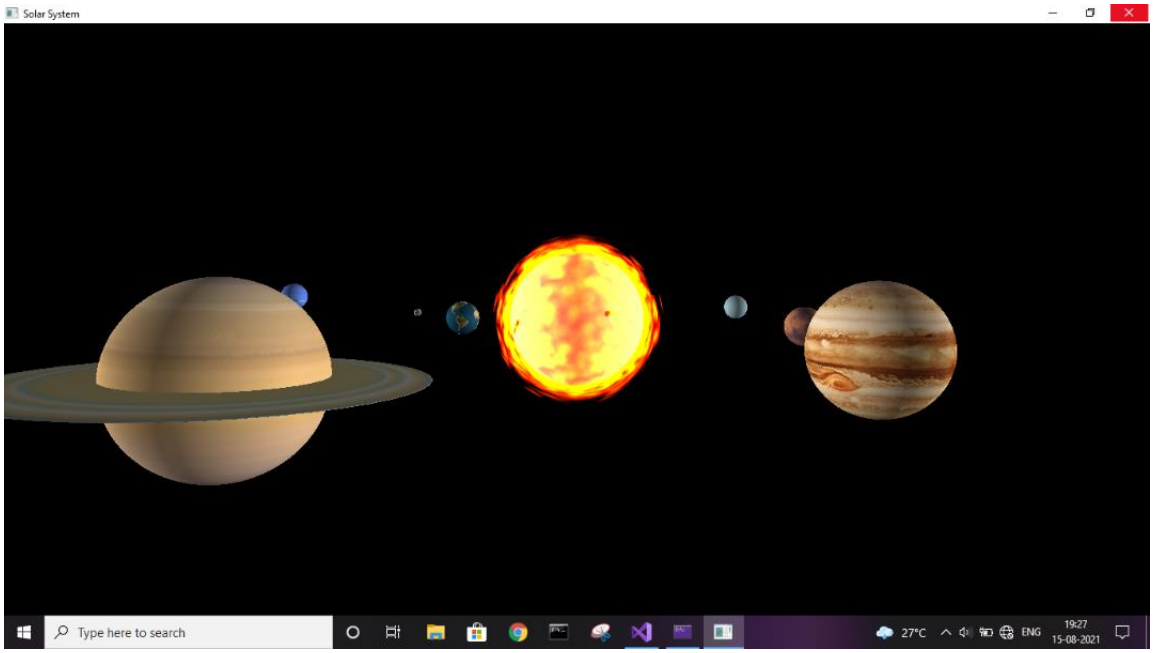

Fig: 5.2 Final view of Solar System

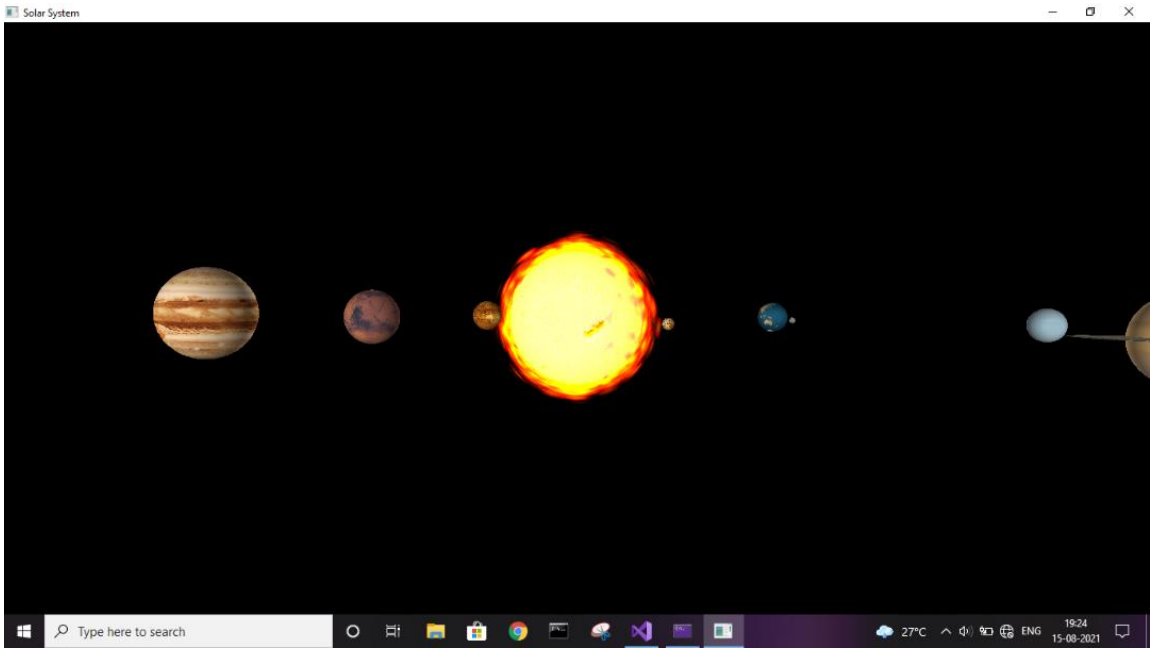

Fig: 5.3 Final view of Solar System

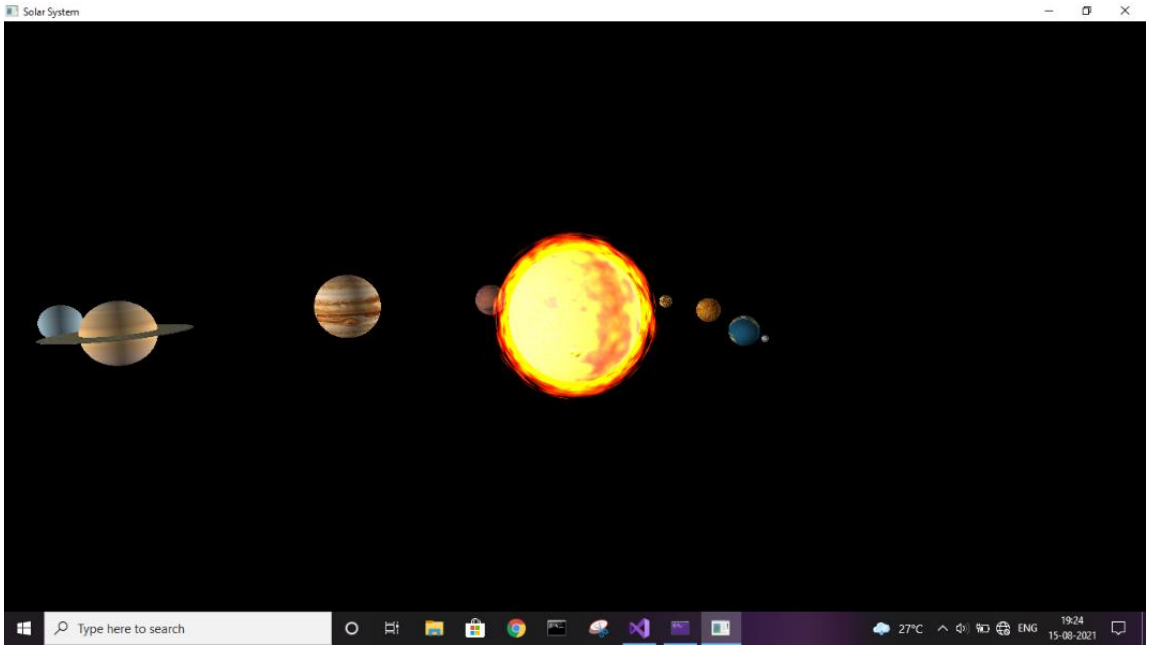

Fig: 5.4 Final view of Solar System

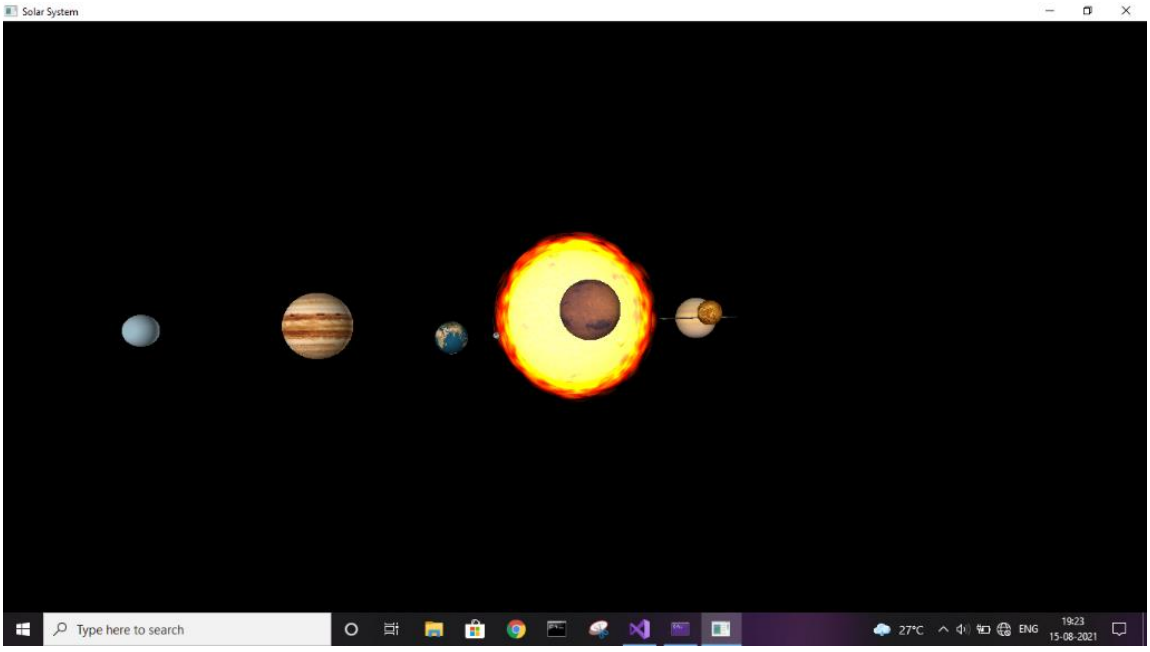

Fig: 5.5 Final view of Solar System

5.2.3 Solar system with revolution of planets:

In this snapshot, we can see sun at the centre and all eight planets revolving around the sun.

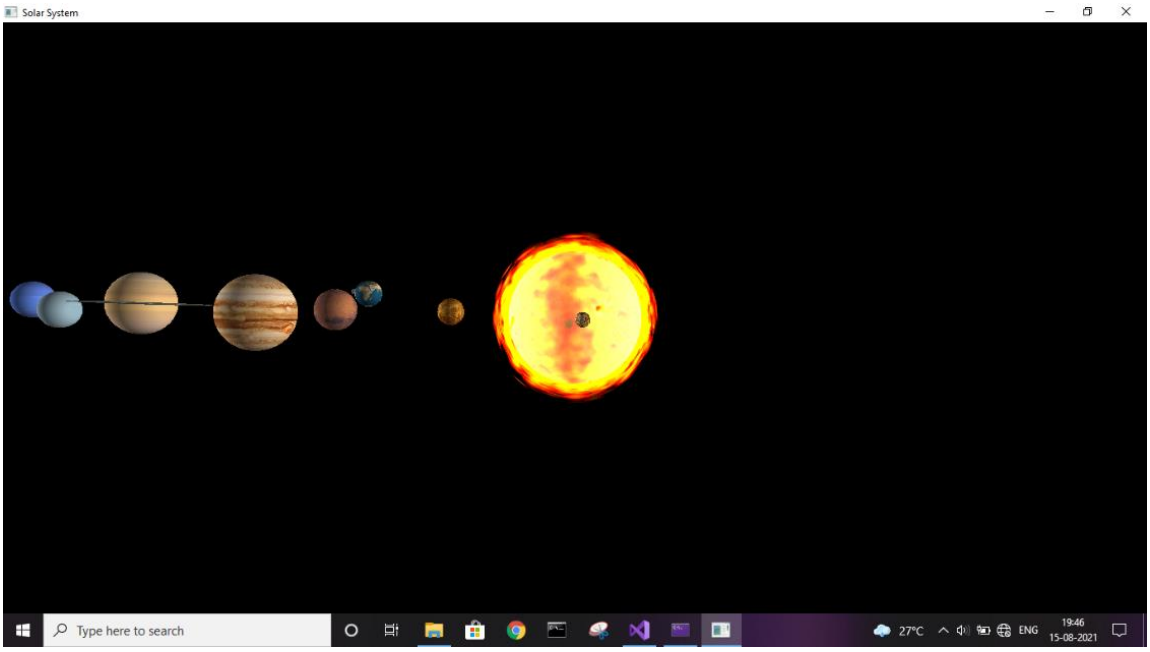

Fig: 5.6 Revolution of planets (Front View)

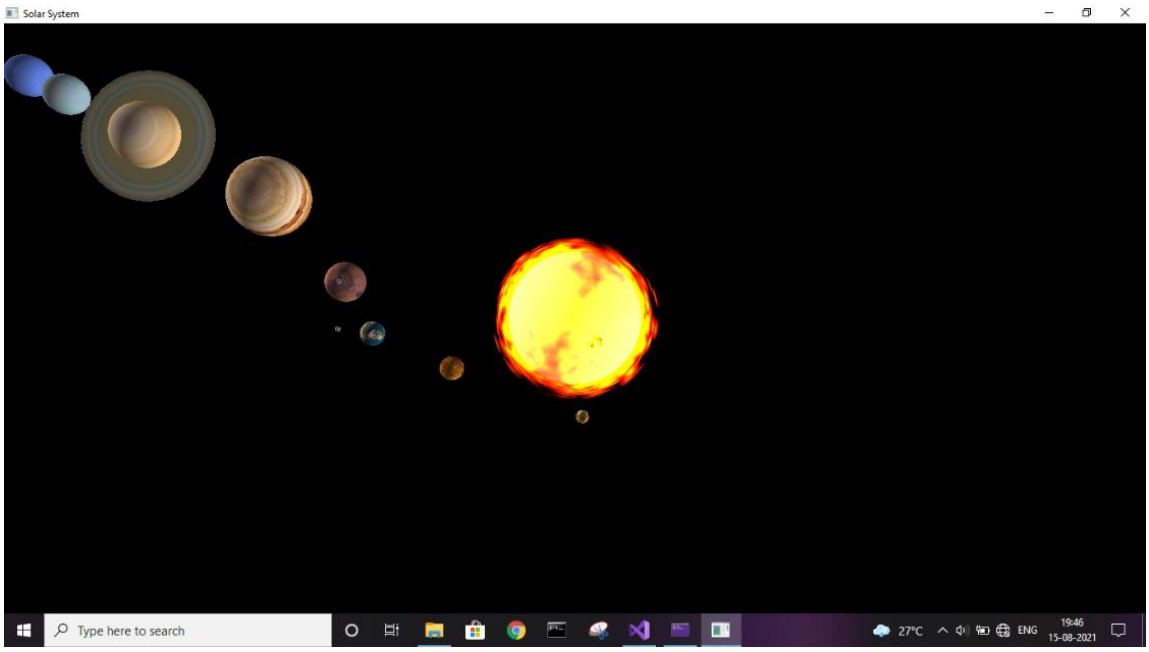

Fig: 5.7 Revolution of planets (Top View)

5.2.4 Viewer positioning angles:

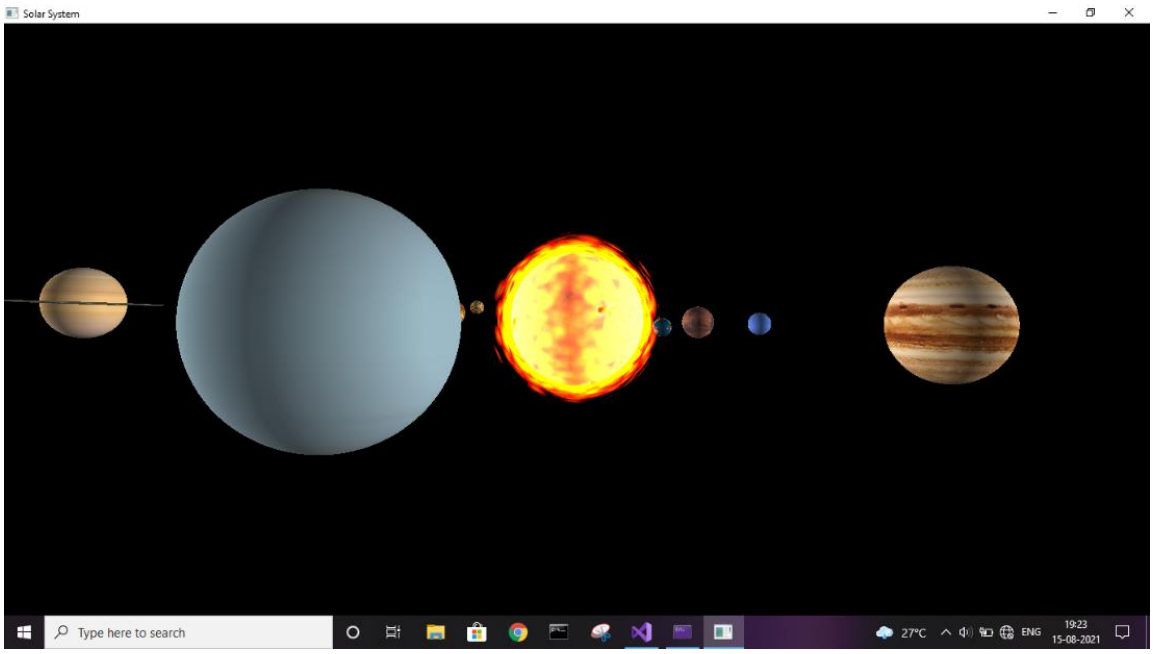

Fig: 5.8 Front View

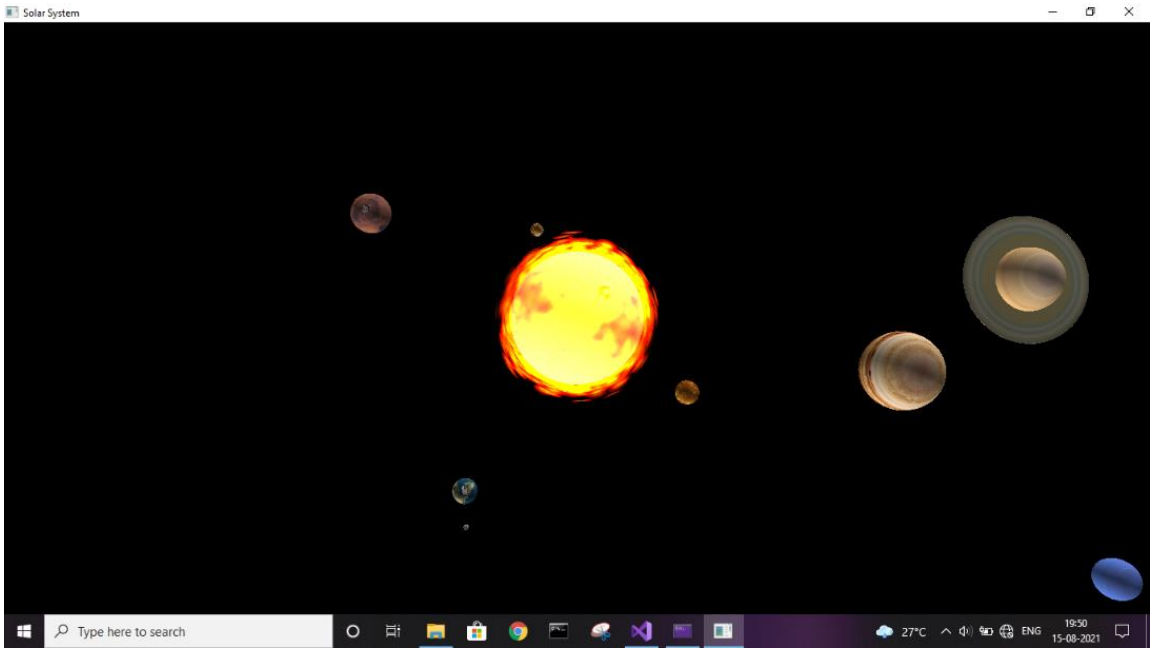

Fig: 5.9 Top View

5.2.5 Blending Function (Enabled/Disabled):

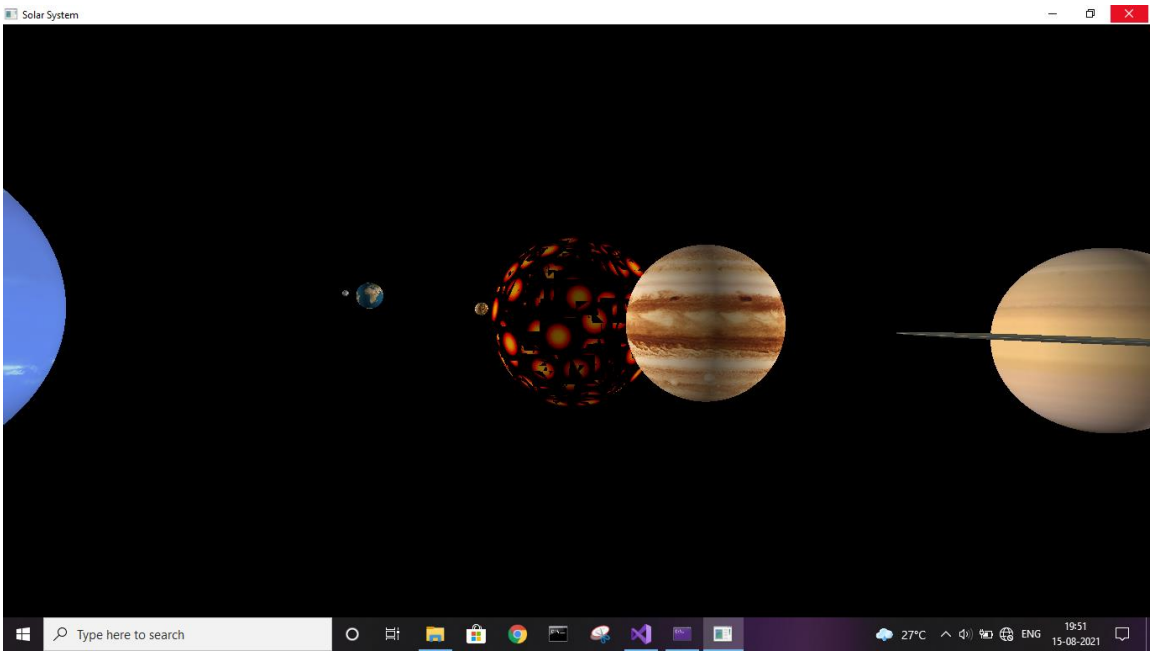

Fig: 5.10 Blending function disabled

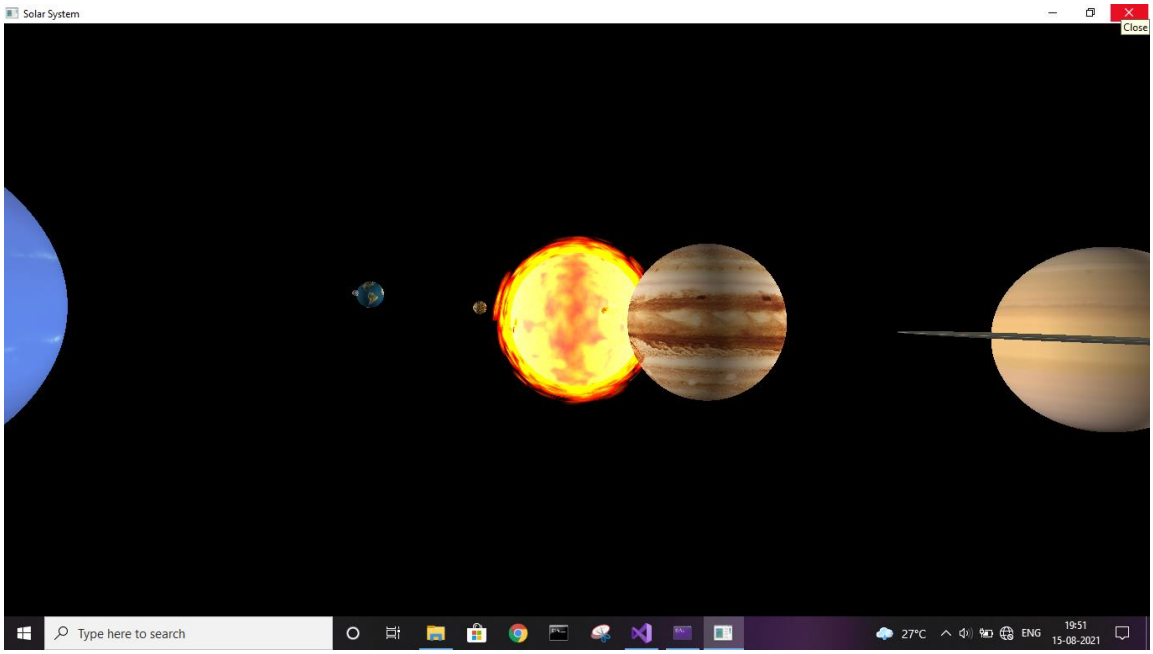

Fig: 5.11 Blending function enabled

5.2.6 Particle System (Enabled and Disabled):

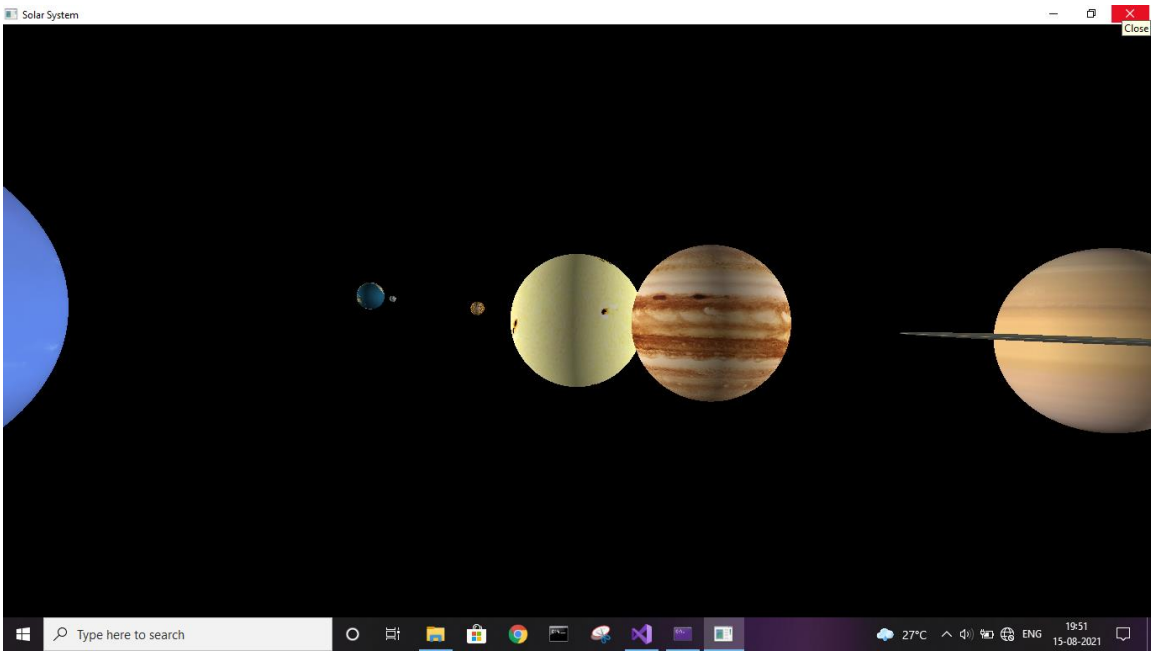

Fig: 5.12 Particle system disabled

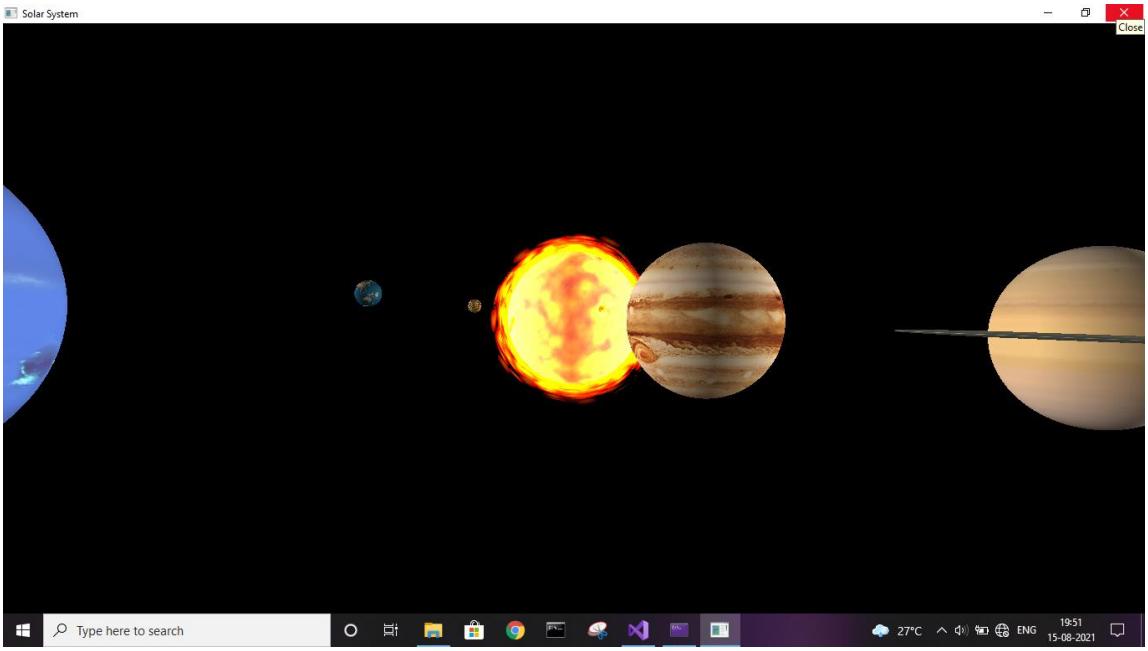

Fig: 5.13 Particle system enabled

5.2.7 Without Texture Mapping:

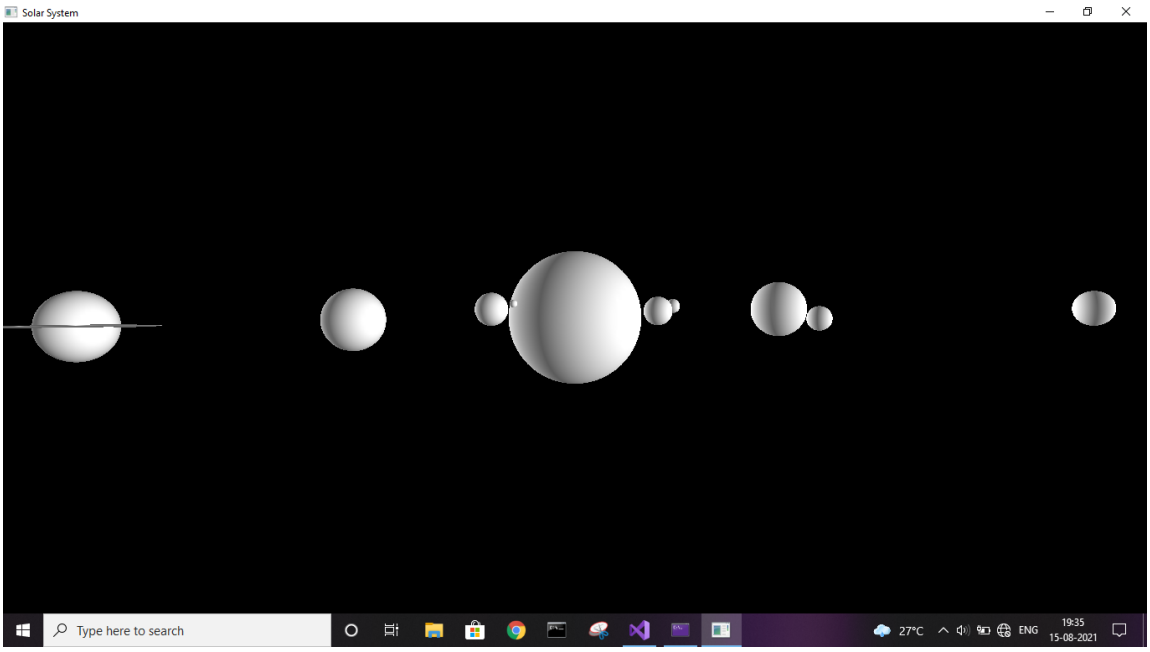

Fig: 5.14 Without texture mapping (Front View)

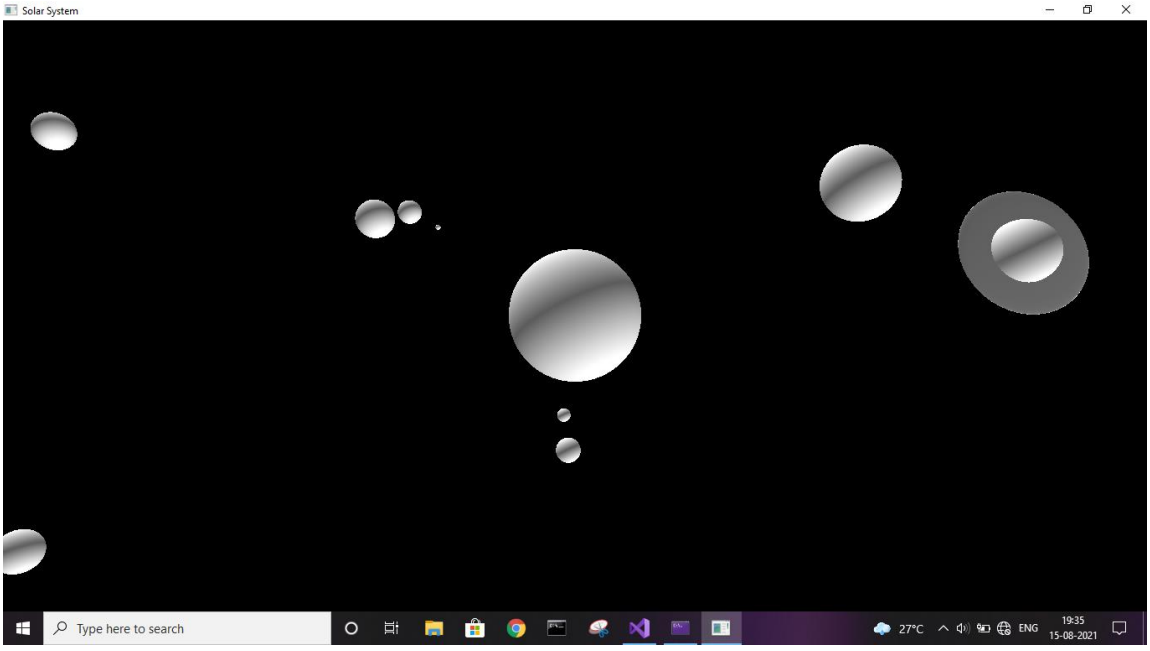

Fig: 5.15 Without texture mapping (Top View)

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENTS

The code we have implemented for our project is working well to the best of our knowledge. In this project the planets and sun act as per the user's command. This project will serve as a delight to the eyes of the night sky watchers.

This project is both informative and entertaining. This project provided an opportunity to learn the various concepts of the subject in detail and provided us a platform to express our creativity and imagination come true.

We found designing and developing this Solar System as a very interesting and learning experience. It helped us to learn about computer graphics, design of Graphical User Interfaces, user interaction handling and screen management.

These are the features that are planned to be supported in the future:

* To make it more user interactive (description of each planet)

*To implement stars and comets

* To improve the looks of the project

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
8. https://learnopengl.com/

Textures:

9. https://www.solarsystemscope.com/textures/

Data:

10. https://theplanets.org/distances-between-planets/
11. http://planetfacts.org/orbital-speed-of-planets-in-order/
12. https://en.wikibooks.org/wiki/GLSL_Programming/GLUT/Textured_Spheres
13. https://en.wikipedia.org/wiki/Solar_rotation

References:

14. https://www.khronos.org/
15. https://www.youtube.com/user/ThinMatrix
16. https://www.youtube.com/user/thebennybox
17. https://www.openal.org/documentation/OpenAL_Programmers_Guide.pdf