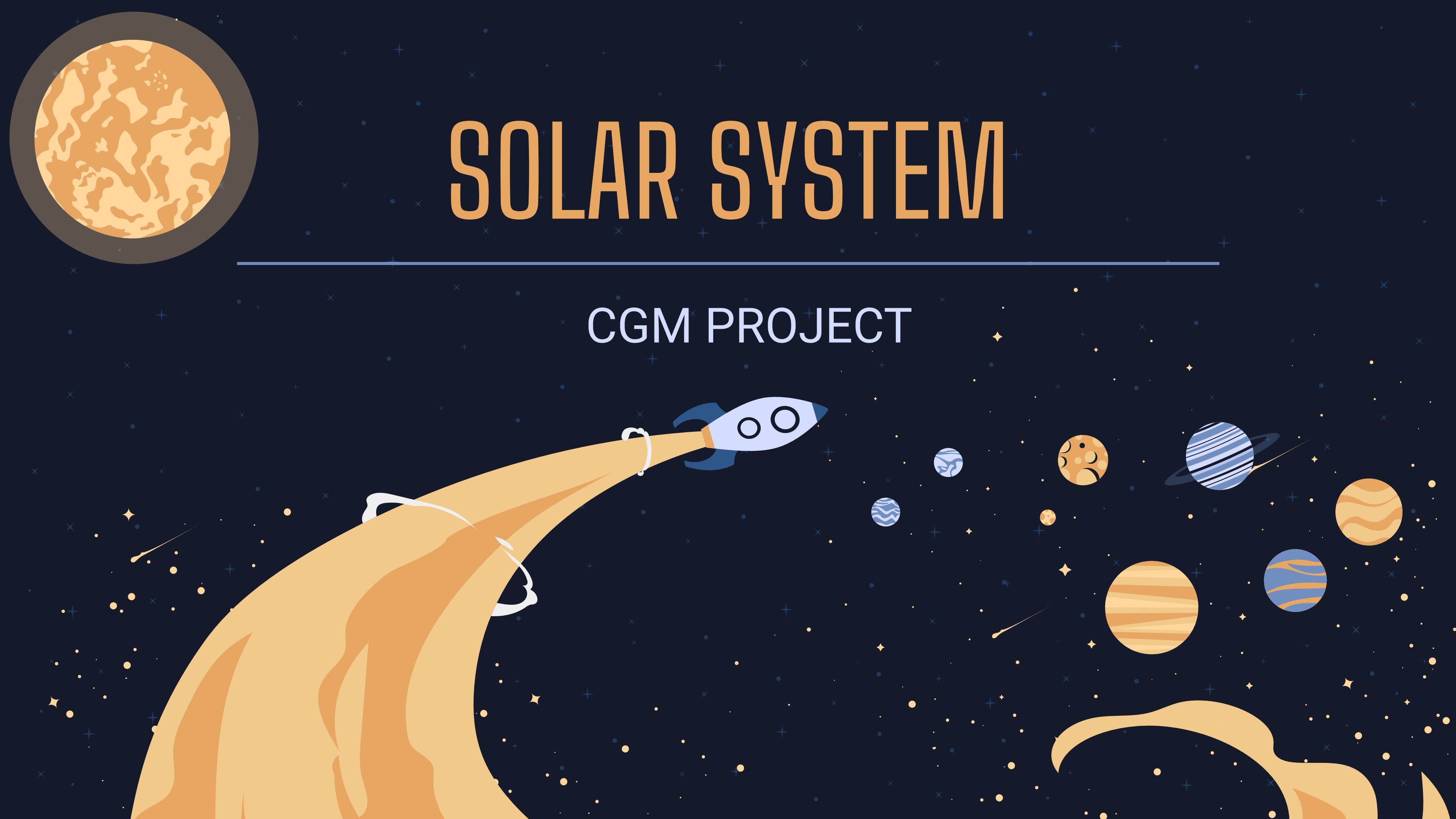
SOLAR SYSTEM

CGM PROJECT



Abstract

Computer graphics is the process of making the design, 2D, 3D, and animation of an object.

This visualization shows the solar system's planetary motion which moves around the sun and remains in each orbit.

We use OpenGL to represent the solar system as a visual.

Problem Motivation

The simulation of the images can't be captured by an actual camera. By using OpenGL, we view the 3d objects and the 3d space that helps to project and capture the frame from any angle in the solar system.

Problem Statement

To write a program in OpenGL that will accurately animate the Solar System which can't be captured by an actual camera precisely.

Problem Solution

By using OpenGL, we view the 3d objects and the 3d space that helps to project and capture the frame from any angle in the solar system.

Requirements

- OpenGL -A cross-platform graphics API that specifies a standard software interface for 3D graphics processing hardware.
- MinGW-A minimal Windows port of the GNU compiler tools, such as GCC, G++, Make, and so on.
- FreeGlut- An open-source alternative to GLUT (OpenGL Utility Toolkit) library which allows the user to create and manage windows.
- Glew - A cross-platform C/C++ library that helps in querying and loading OpenGL extensions.
- VSCode Editor - A streamlined code editor with support for development operations like debugging, task running, and version control.

Libraries

```
#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <math.h>
#include <time.h>
#include <windows.h>
```

How to execute the File:

```
g++ SolarSystem.cpp -o SolarSystem lopengl32 lglew32 -lfreeglut -lglu32
```

An executable file is created after the successful execution of the code

How to run the .exe File:

```
Run ./SolarSystem.exe
```

Details about Files

- ➢ SolarSystem.cpp:

This is the main driver for the program. It starts the program and does most of the work.

- ➢ SolarSystem.cpp and SolarSystem.exe:

When these files work together they generate a sphere in OpenGL. These spheres are used for a couple of different objects in the Solar System, such as each planet and
the orbits of asteroids.

- ➢ GLUT-OpenGL plugins:

Add glut libraries from the extracted folder for the required scenarios in the bin and Lib Folder.

Additional Functions used for OpenGL

- glutInit: It is used to initialize the GLUT library
- glEnable(GL_COLOR MATERIAL): It enables to plug in colors you want to enter into it
- glBegin(GL_QUADS): Quadruples of vertices used to draw quadrilaterals
- glPushMatrix and glPopMatrix : To push and pop the current matrix stack
- glVertex: It specifies a vertex
- glLight: This sets light source parameters
- glMaterialfv: It specifies material parameters for the lighting model
- glutTimerFunc: It registers a timer callback to be triggered in a specified number of millisecond
- glutMainLoop: It enters the GLUT event processing loop
- glutPostRedisplay: It marks the current window as needing to be re-displayed

Material

Material of the solar system based on the discussion as needed for design, there are:

1. The composition of the solar system Displays information about the sun and the order of the planets and their orbital trajectory
2. Various planets Displays the names of the planets in order
3. The size of the planet Displaying diameter comparison by the actual size of the planet and visualization
4. Distance planet to the sun Displaying diameter comparison by the actual distance of the planet and visualization
5. The revolution speed of each planet Revolution speed comparison by the actual speed of the planet and the visualization
6. The orbit of each planet Tracks Display every planetary orbit

Research Method

In this project, there are 2 types of models that will be used to illustrate the planet. The first model is a simple sphere that represents the 3D shape for most of the planets. The second model is a sphere with a circular ring surrounding the object. The model is used on planets such as Neptune, Saturn, and Uranus

Stages of making solar system visualization:

1. Declare all the attributes that are needed for the translation and rotation functions.
2. Set the speed of rotation of each planet, because the speed of each planet to make one revolution is different
3. We had created two separate paths for planets and asteroids.
4. Create the sun with a solid sphere, set the size, and position it in the center point (0,0,0) so that the sun becomes the center of planets that will be surrounding them.

5. Create the planets Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune with a solid sphere, set the size of each planet, and the planet's position in accordance with the ratio of the actual distance.
6. Set the revolution of each planet.
7. To generate the final directional light effect, it will require the ambient, diffuse and specular reflection summation.
8. Create the orbit of each planet, with torus (ring), set torus in the center, so the planets will move around the sun, and still remain in orbit.
9. Adjust the lighting to give a 3D effect

Size of planet

The real data about the diameter of the planet, we got from NASA's official website. We did a comparison scale of diameter for use in OpenGL and applied the same formula on each planet, the formula is a real diameter divided by 5, to make the size to be small. So the scale is 5:1

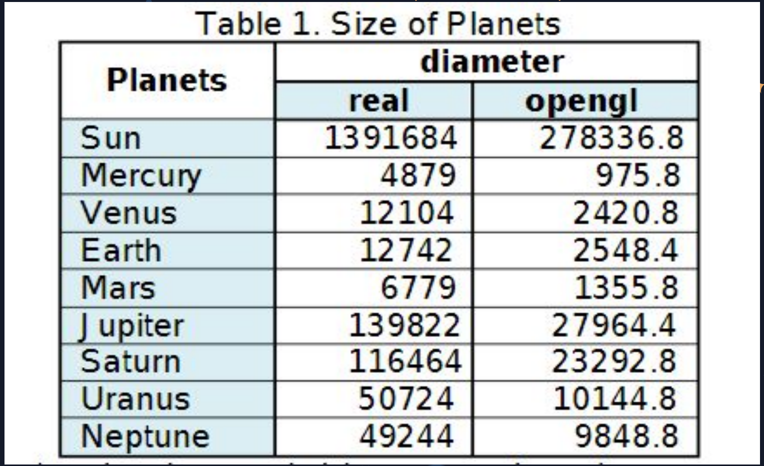


Table 1. Size of Planets

Planets	diameter	
	real	opengl
Sun	1391684	278336.8
Mercury	4879	975.8
Venus	12104	2420.8
Earth	12742	2548.4
Mars	6779	1355.8
J upiter	139822	27964.4
Saturn	116464	23292.8
Uranus	50724	10144.8
Neptune	49244	9848.8

Speed of Each Planet Revolution

The real data about the distance of the planet, we got from NASA’s official website. We did a comparison scale of distance for use in OpenGL and applied the same formula on each planet, the formula is visual planet distance plus a real diameter of the sun, to make the right position in coordinate. Because the distance is measured from the outer surface of the sun. So the scale is 1000:1.

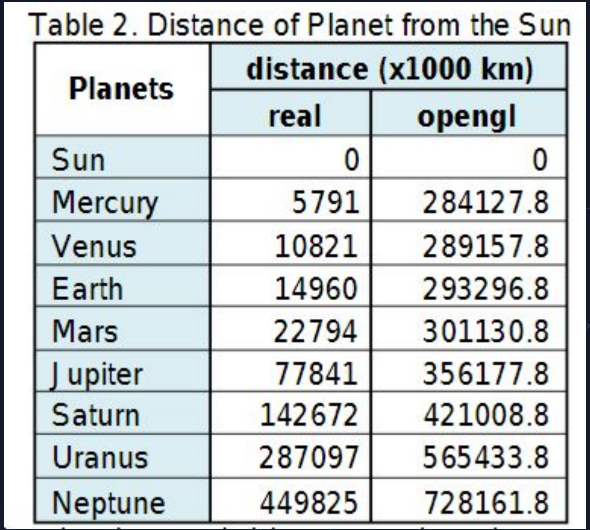


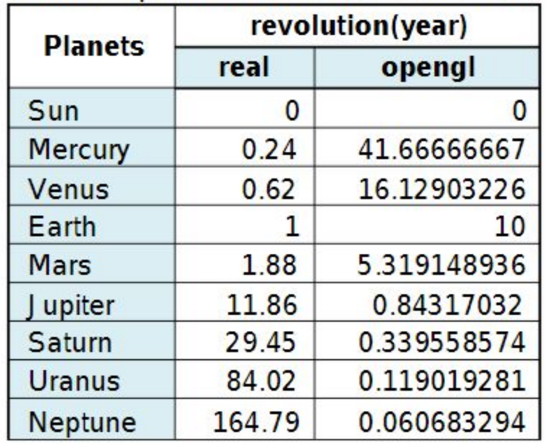
Table 2. Distance of Planet from the Sun

Planets	distance (x1000 km)	
	real	opengl
Sun	0	0
Mercury	5791	284127.8
Venus	10821	289157.8
Earth	14960	293296.8
Mars	22794	301130.8
Jupiter	77841	356177.8
Saturn	142672	421008.8
Uranus	287097	565433.8
Neptune	449825	728161.8

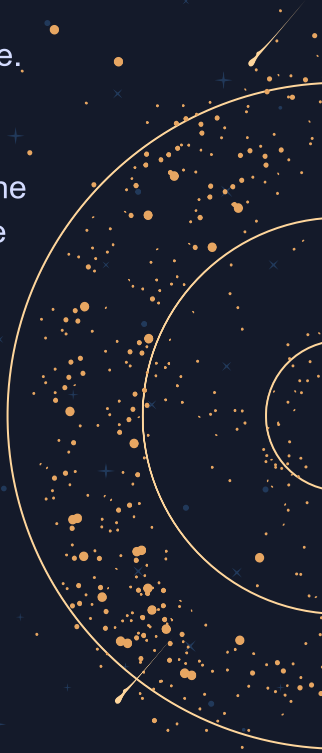
Distance of Planet from the Sun

The real data about the speed of each planet’s revolution was obtained from the Nasa official website. We did a comparison scale of speed or revolution for use in OpenGL and applied the same formula on each planet. In OpenGL, the larger number has faster movements. Then we did a comparison of the speed of the earth, with another planet. Each velocity is divided by the speed of the earth, then the planet which has a faster revolution than the earth will have a larger number than the Earth. Then we multiply with 10, to make a movement that can be seen at OpenGL. So the scale is 1:10.

Table 3. Speed of Each Planet Revolution



Planets	revolution(year)	
	real	opengl
Sun	0	0
Mercury	0.24	41.66666667
Venus	0.62	16.12903226
Earth	1	10
Mars	1.88	5.319148936
J upiter	11.86	0.84317032
Saturn	29.45	0.339558574
Uranus	84.02	0.119019281
Neptune	164.79	0.060683294

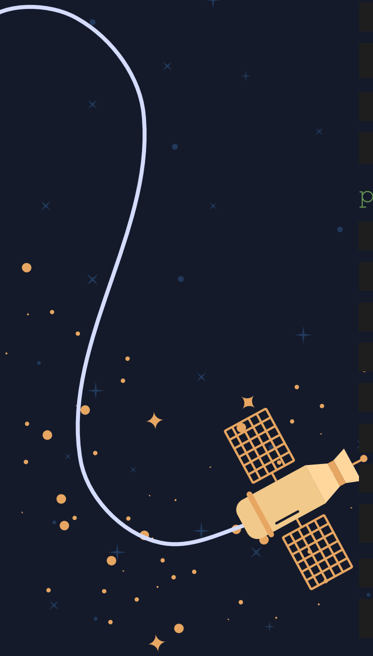


Code Snippets

```
int main(int argc, char **argv)
{
 glutInit(&argc, argv);                    //glutInit is used to initialize the GLUT
library.
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
 glutInitWindowPosition(0, 0);             //initial window position
 glutInitWindowSize(700, 700);             //initial window size
 glutCreateWindow("Solar System");         //creates a window named "solar System"
 Light_initialize();                       //ambient diffuse and specular
 initialize_display();
 glutDisplayFunc(callback_planets);  glutTimerFunc(25, change_axis, 0);
 glutMainLoop();
 return 0;
}
```

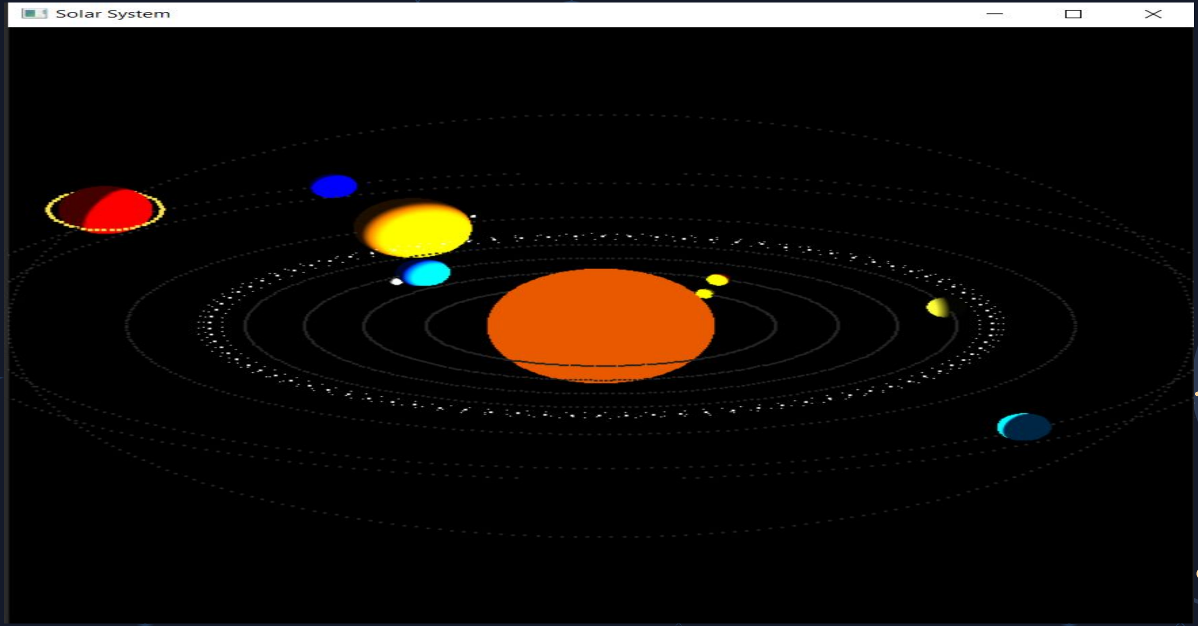
```
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);   //glClear function sets the bitplane area of the
window to values previously selected
 display_background ();
 display_orbit ();
 glLoadIdentity (); //Replaces the current matrix with the identity matrix
 glPushMatrix (); //Pushes the current matrix stack down by one, duplicating the current matrix
 glEnable (GL_DEPTH_TEST);  //Enables various OpenGL graphics capabilities.
 glEnable (GL_COLOR_MATERIAL);
 glPushMatrix (); //Pushes the current matrix
 glColor3f (0.7, 0.5, 0.0);
 glScalef (scale_x, scale_y, scale_z);
 glLightfv (GL_LIGHT7, GL_POSITION,  temp_color1 );
 glMaterialfv (GL_FRONT_AND_BACK, GL_EMISSION,  Yellow_color );
 glutSolidSphere (1, 50, 50);
 glMaterialfv (GL_FRONT_AND_BACK, GL_EMISSION,  Black_color );
 glPopMatrix ();
}

glPushMatrix ();
 glRotatef (Mercury_rotate , 0.0, 1.0, -0.5); //Counter-clockwise rotation of angle degrees about the vector
 glTranslatef (1.5, 0.0, 0.0); //Multiplies the current matrix by a translation matrix
 glColor3f (1.0, 0.9, 0.0); //Used to give each vertex its own color
 glScalef (0.08, 0.08, 0.08); //A general scaling along the x, y, and z axes
 glutSolidSphere (1, 50, 50);
 glPopMatrix ();       //Pops the current matrix
```

```
glPushMatrix(); //Pushes the current matrix
glColor3f(3.30, 3.30, 3.30);
glRotatef(63, 1.0, 0.0, 0.0); //Counter-clockwise rotation of angle degrees about the vector
int j = 0, i = 0, rotate_d = 90;
float length = 2;
float scale_temp[4] = {3.3, 3.4, 3.35, 3.2};
for (j = 0; j < 4; j++){
  glPushMatrix();
  length -= 0.3;
  glPointSize(length); //Specifies the rasterized diameter of points
  glScalef(scale_temp[j], scale_temp[j], scale_temp[j]);
  glBegin(GL_POINTS); //Delimit the vertices that define a primitive or group of like primitives.
  double rotation_deg2 = 0.0 - Astroid_rotate, a = (2 * (3.14)) / rotate_d;
  for (i = 0; i < rotate_d; i++){
    glVertex2d(cos(rotation_deg2), sin(rotation_deg2));
    rotation_deg2 += a;
  }
  rotate_d += 10
  glEnd();
  glPopMatrix(); //Pops the current matrix
 }
  glPopMatrix(); //Pops the current matrix
```



Experimental Results

The first appearance is an animation of planet movement and some asteroids, they move around the sun automatically, and it also gives general information about the solar system



Conclusion

❖ To conclude, this project shows how computer graphics could simulate some pictures that can’t be captured easily with the actual camera. By having objects and using some computer graphic techniques, the final result portrays the solar system generally.

❖ All the planets are set to some specified color and time frame to rotate around the sun which looks realistic. Here we set some asteroids to make them realistic.

❖ Subsequently, all the computer graphic methods have made the final frame rendered to be more alive and looked realistic.

THANK YOU