VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELGAUM-590018

A Mini-Project Report
On

"Working of Satellite"

A Mini-project report submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Engineering in Computer Science and Engineering** of Visvesvaraya Technological University, Belgaum.

Submitted by:

MUBASHIR PARWAZ DAR(1AM19CS245)
OM PRAKASH JHA(1AM19CS133)
THEJAS DHANESH POONTHOTTATHIL(1AM19CS234)
ZAIN UD DIN(1AM19CS249)

Under the Guidance of:

Mrs. Veena Bhatt
(Assistant Professor, CSE)

Department of Computer Science and Engineering
AMC Engineering College

18th K.M, Bannerghatta Main Road, Bangalore-560 083

2021-2022

AMC Engineering College,

18th K.M, Bannerghatta Main Road, Bangalore-560 083

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the mini-project work entitled **"Working of Satellite"** has been successfully carried out by **MUBASHIR PARWAZ DAR(1AM19CS245) , OM PRAKASH JHA(1AM19CS133), THEJAS DP(1AM19CS234), ZAIN UD DIN(1AM19CS249) bonafide** students of **AMC Engineering College** in partial fulfillment of the requirements for the award of degree in **Bachelor of Engineering in Computer Science and Engineering** of **Visvesvaraya Technological University, Belgaum** during academic year 2021-2022 . It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of project work for the said degree.

Guide:

Mrs. Veena Bhatt	**Dr. NAGARAJA R**	**Dr. GIRISHA C**
Assistant Professor, CSE	**Professor & Head, CSE**	**Principal, AMCEC**

Examiners: **Signature with Date**

1.

2.

ACKNOWLEDGEMENT

It gives us immense pleasure to present before you our project titled **'WORKING OF A SATELLITE USING OPENGL'**. The joy and satisfaction that accompany the successful completion of any task would be incomplete without the mention of those who made it possible. We are glad to express our gratitude towards our prestigious institution **AMC ENGINEERING COLLEGE** for providing us with utmost knowledge, encouragement and the maximum facilities in undertaking this project.

We sincerely acknowledge the guidance and constant encouragement of our mini-project guide, **Mrs. Veena Bhatt, Assistant Professor, Department Of Computer Science Engineering.**

We express our deepest gratitude and special thanks to **Dr. NAGARAJA R**, **Professor & Head, Department Of Computer Science Engineering**, for all his guidance and encouragement.

We wish to express sincere thanks to our respected chairman **Dr. K.R. PARAMAHAMSA** and beloved principal **Dr. GIRISHA C** for all their support.

MUBASHIR PARWAZ DAR(1AM19CS245)
OM PRAKASH JHA(1AM19CS133)
THEJAS DP(1AM19CS234)
ZAIN UD DIN(1AM19CS249)

ABSTRACT

- Main aim of this Mini Project is to illustrate the concepts of working of a Satellite in OpenGL.
- A S**atellite** is an object which has been placed into orbit by human endeavor. Such objects are sometimes called **artificial satellites** to distinguish them from natural satellites such as the Moon.
- Satellites are used for a large number of purposes. Common types include military and civilian Earth observation satellites, communications satellites, navigation satellites, weather satellites, and research satellites.
- This pushed the entire network into a 'congestion collapse' where most packets were lost and the resultant throughput was negligible.
- We have used input devices like mouse and key board to interact with program.
- We have also used SolidCube for forming a complete network setup which help to understand concept of Congestion Control very well.
- To differentiate between objects we have used different colors for different objects.
- We have added menu which makes the program more interactive.
- In this project we have used a small SolidCube to represent a data, which travels as data transfer from source to destination.
- We have used font family for indicating the name of objects as we can see in this project.

CONTENTS

CERTIFICATE **(i)**

ACKNOWLEDGEMENT **(ii)**

ABSTRACT **(iii)**

1. INTRODUCTION **1**

1.1 Introduction to Computer graphics
1.2 History of Computer graphics
1.3 Applications of Computer graphics
1.4 Open Graphics Library (OpenGL)
1.5 OpenGL Utility Library (GLU)
1.6 OpenGL Utility Toolkit (GLUT)
1.7 OpenGL Primitives
1.8 Introduction to Graph Coloring

2. SYSTEM SPECIFICATION **9**

2.1 Software requirements
2.2 Hardware requirements

3. Working of Satellite **10**

3.1 Interaction with the program **10**

4. SYSTEM DESIGN AND IMPLEMENTATION **11**

4.1 Introduction
4.2 Initialization
4.3 Flow of control
4.4 OpenGL APIs used/Built-in functions
4.5 Pseudo Codes
4.6 Source Code

5. RESULTS AND DISCUSSIONS **25**

5.1 Screenshots

CONCLUSION **29**

REFERENCES **30**

CHAPTER 1

INTRODUCTION

1.1 Introduction to Computer Graphics

Computer graphics are pictures and films created using computers. Usually, the term refers to computer-generated image data created with help from specialized graphical hardware and software. It is a vast and recent area in computer science. The phrase was coined in 1960, by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, though sometimes erroneously referred to as computer-generated imagery (CGI).Some topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modelling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others. The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. Computer graphics is responsible for displaying art and image data effectively and meaningfully to the user. It is also used for processing image data received from the physical world. Computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design generally [1].

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: [1]

- the representation and manipulation of image data by a computer
- the various technologies used to create and manipulate images
- the sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content, see study of computer graphics

Today, computer graphics is widespread. Such imagery is found in and on television, newspapers, weather reports, and in a variety of medical investigations 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, theses", and other presentation material.

Many tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: two dimensional (2D), three dimensional (3D), and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used. 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 (time) component". [5]

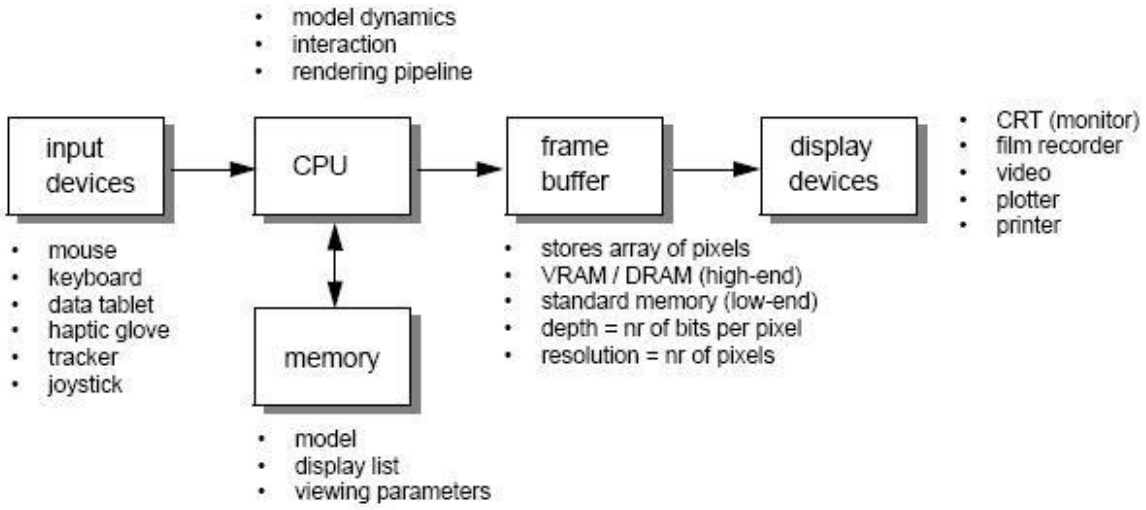


Figure 1.1 Architecture of Computer Graphics with its components

1.2 History of Computer Graphics

The phrase Computer Graphics was coined in 1960 by William Fetter, a graphic designer for Boeing. The field of Computer Graphics developed with the emergence of computer graphics hardware. Early projects like the Whirlwind and SAGE projects introduced the CRT as a viable display and interaction interface and introduced the light pen as an input device. [1]

Also, in 1961 another student at MIT, Steve Russell, created the first video game, Spacewar! E.Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-gyro gravity attitude control system" in 1963. In this computer-generated film, Zajac showed how the attitude of a satellite could be altered as it orbits the

Earth. Many of the most important early breakthroughs in computer graphics research occurred at the University of Utah in the 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.

Graphics and application processing were increasingly migrated to the intelligence in the workstation, rather than continuing to rely on central mainframe and mini-computers. 3D graphics became more popular in the 1990s in gaming, multimedia and animation. Computer graphics used in films and video games gradually began to be realistic to the point of entering

the uncanny valley. Examples include the later *Final Fantasy* games and animated films like *The Polar Express*.[1]

1.3 Applications of Computer graphics

The development of computer graphics has been driven both by the needs of the user community and by advances in hardware and software. The applications of computer graphics are many and varied. We can however divide them into four major areas [6]

- Design: Professions such as engineering and architecture are concerned with design. Today, the use of interactive graphical tools in CAD, in VLSI circuits, characters for animation have developed in a great way.
- Simulation and animation: One of the most important uses has been in pilots' training. Graphical flight simulators have proved to increase safety and reduce expenses. Simulators can be used for designing robots, plan its path, etc. Video games and animated movies can now be made with low expenses.

1.4 Open Graphics Library (OpenGL)

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that are used to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined hardware-independent interface to be implemented on many different hardware platforms. [2]

These are certain characteristics of OpenGL:

- OpenGL is a better documented API.
- OpenGL is much easier to learn and program.
- OpenGL has the best demonstrated 3D performance for any API.

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it's possible for the API to be implemented entirely in software, it's designed to be implemented mostly or entirely in hardware.

In addition to being language-independent, OpenGL is also platform-independent. The specification says nothing on the subject of obtaining, and managing, an OpenGL context, leaving this as a detail of the underlying windowing system. For the same reason, OpenGL is purely concerned with rendering, providing no APIs related to input, audio, or windowing.

OpenGL is an evolving API. New versions of the OpenGL specification are regularly released by the Khronos Group, each of which extends the API to support various new features. In addition to the features required by the core API, GPU vendors may provide additional functionality in the form of *extensions*. Extensions may introduce new functions and new constants and may relax or remove restrictions on existing OpenGL functions. Vendors can use extensions to expose custom APIs without needing support from other vendors or the Khronos Group as a whole, which greatly increases the flexibility of OpenGL. All extensions are collected in, and defined by, the OpenGL Registry. [2]

1.5 OpenGL Utility Library (GLU)

The OpenGL Utility Library (GLU) was a computer graphics library for OpenGL.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.

Among these features are mapping between screen- and world-coordinates, generation of texture mipmaps, drawing of quadric surfaces, NURBS, tessellation of polygonal primitives, interpretation of OpenGL error codes, an extended range of transformation routines for setting up viewing volumes and simple positioning of the camera, generally in more human-friendly terms than the routines presented by OpenGL. It also provides additional primitives for use in OpenGL applications, including spheres, cylinders and disks.

All GLU functions start with the glu prefix. An example function is gluOrtho2D which defines a two-dimensional orthographic projection matrix.

1.6 OpenGL Utility Toolkit (GLUT)

The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres and the Utah teapot.

GLUT was written by Mark J. Kilgard, author of OpenGL Programming for the X Window System and The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics, while he was working for Silicon Graphics Inc. [2]

The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is cross-platform) and to make learning OpenGL easier. Getting started with OpenGL programming while using GLUT often takes only a few lines of code and does not require knowledge of operating system–specific windowing APIs. All GLUT functions start with the glut prefix , for example, glutPostRedisplay marks the current window as needing to be redrawn. [5]

The toolkit supports:

- Multiple windows for OpenGL rendering and call back driven event processing
- Sophisticated input devices
- An 'idle' routine and timers
- A simple, cascading pop-up menu facility
- Utility routines to generate various solid and wire frame objects
- Support for bitmap and stroke fonts
- Miscellaneous window management functions

Some of the more notable limitations of the original GLUT library by Mark Kilgard include:

The library requires programmers to call glutMainLoop(), a function which never returns. This makes it hard for programmers to integrate GLUT into a program or library which wishes to have control of its own event loop. A common patch to fix this is to introduce a new

function,called glutCheckLoop() (macOS)or glutMainLoopEvent() (FreeGLUT/OpenGLUT), which runs only a single iteration of the GLUT event loop. Another common workaround is to run GLUT's event loop

- In a separate thread, although this may vary by operating system, and also may introduce synchronization issues or other problems: for example, the macOS GLUT implementation requires that glutMainLoop() be run in the main thread. [2]
- The fact that glutMainLoop() never returns also means that a GLUT program cannot exit the event loop. FreeGLUT fixes this by introducing a new function, glutLeaveMainLoop().
- The library terminates the process when the window is closed; for some applications this may not be desired. Thus, many implementations include an extra callback, such as glutWMCloseFunc().

1.7 OpenGL Primitives

OpenGL supports two classes of primitives:

- Geometric Primitives: Geometric primitives are specified in the problem domain and include points, line segments, polygons, curves and surfaces.
- Image (Raster) Primitives: Raster primitives, such as arrays of pixels pass through a separate parallel pipeline on their way to the frame buffer.

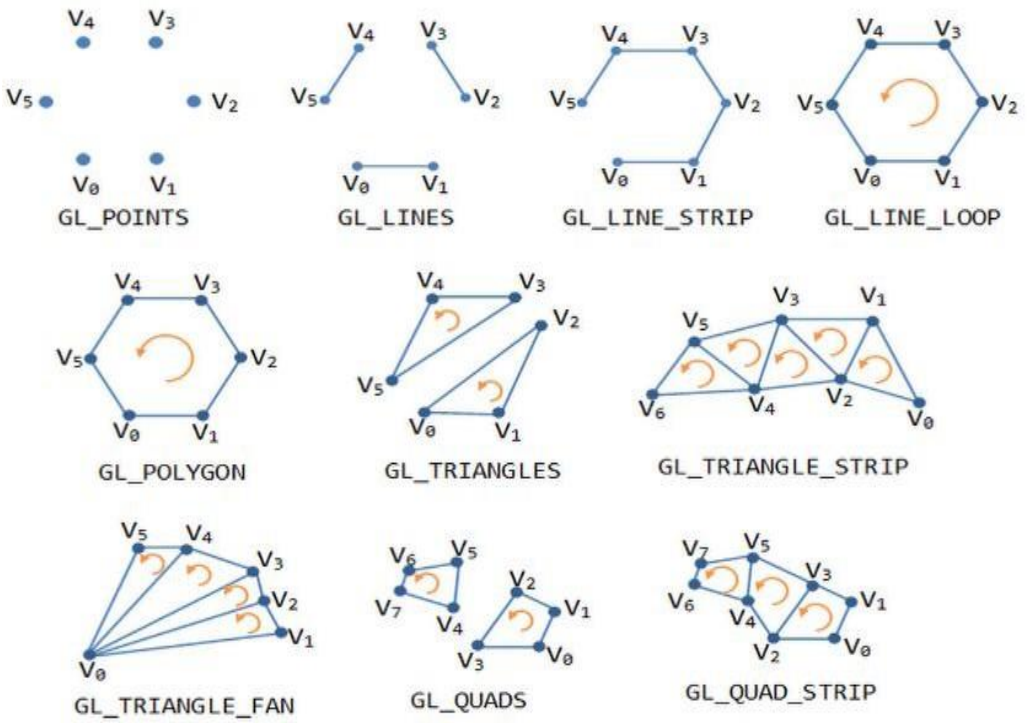


Figure 1.2 OpenGL Geometric Primitives

1.8 Introduction to Graph Coloring

In graph theory, a graph coloring is an assignment of colors to certain objects in a graph subject to certain constraints. The simplest form of graph coloring, called the proper or legal vertex coloring, is to assign colors to the vertices of a graph such that no two adjacent vertices share the same color. Similarly, a proper edge coloring assigns colors to the edges such that no two incident edges share the same color, while a proper face coloring of a planar graph assigns colors to the faces or regions such that no two adjacent faces share the same color. Typically, other coloring problems can be transformed into a vertex coloring version. For example, an edge coloring of a graph is just a vertex coloring of its adjoint graph, while a face coloring of a planar graph is just a vertex coloring of -its planar dual graph. When used without any further specification, a coloring of a graph is always assumed to be a proper vertex coloring. However, non-vertex coloring problems are usually stated and studied as they are to keep things in their perspective. Graph coloring problem has found a number of applications such as scheduling, compiler allocation, frequency assignment and pattern matching.

A proper coloring of a graph using at most k colors is called a k-coloring. Clearly, finding a k-coloring of a graph is equivalent to the problem of partitioning the vertices into k or fewer independent sets. The least number of colors needed to color a graph is called its chromatic number χ. A graph is called k-colorable if there exists a k-coloring of the graph and is called k-chromatic if its chromatic number is exactly k.

Theorem 1: Let G = (V, E) be a graph. We have

(i) $\chi(G) = 1$ if and only if G is totally disconnected;

(ii) $\chi(G) \geq 3$ if and only if G is not bipartite, or equivalently, has an odd cycle;

(iii) $\chi(G) \geq \omega(G)$, where $\omega(G)$ is the order of a largest clique in G;

(iv) $\chi(G) \leq 4(G) + 1$, where 4(G) is the largest degree over all vertices;

(v) (Brook's Theorem) $\chi(G) \leq 4(G)$ if G is not Kn or C2n+1 where Kn is the complete graph with n vertices and C2n+1 is the odd cycle with 2n + 1 vertices.

$\chi(G) \leq 4$ for any planar graph G. Given a k-colorable graph G = (V, E), finding a k-coloring for G is solvable in polynomial time for k = 2 but NP-hard for $k \geq 3$. The decision version of graph covering problem, i.e., whether or not is there a k-coloring, is one of Karp's 21 NP-complete problems. As shown by Garey et al. [9], it remains to be NP-complete even on planar graphs with node degree at most 4 although it is trivial for $k \geq 4$ on planar graphs due to the four-color theorem.. Johnson [12] showed that a version of the greedy algorithm gives an O(n/ log n)-approximation algorithm for k-colorings where n is the number of vertices in the graph. Wigderson [21] developed an algorithm to find a O(√ n)-coloring for any 3-colorable graph in polynomial time, which can be extended to find O(n 1−1/(k−1))-colorings for general k. Blum [3] provided a combinatorial algorithm for coloring a 3-colorable graph

(vi) using O (n 3/8 log8/5 n) colors, which can be generalized to color a k-colorable graph with O(n 1−1/(k−4/3) log8/5 n) colors. Karger et al. [15] presented a semidefinite programming based O(n 1/4 log1/2 n)-coloring for 3-colorable graphs and O(n 1−3/(k+1) log1/2 n)-coloring for k-colorable graphs. Blum and Karger [4] combined the techniques in Blum [3] and Karger et al. [15] and improved the bound to O˜(n 3/14) for 3- colorable graphs where the notation O˜ is used to hide lower-order multiplicative terms, such as log n. So far, the best known approximation for 3-colorable graphs is due to Arora et al. [1], no polynomial time algorithm can approximate the chromatic number of a graph to within a ratio of n unless P = NP. Feige and Kilian [8] and 2 H˙astad [11] showed that approximating the chromatic number to within n 1−δ for any δ > 0 would imply NP = RP where RP is the class of probabilistic polynomial time algorithms making one sided error

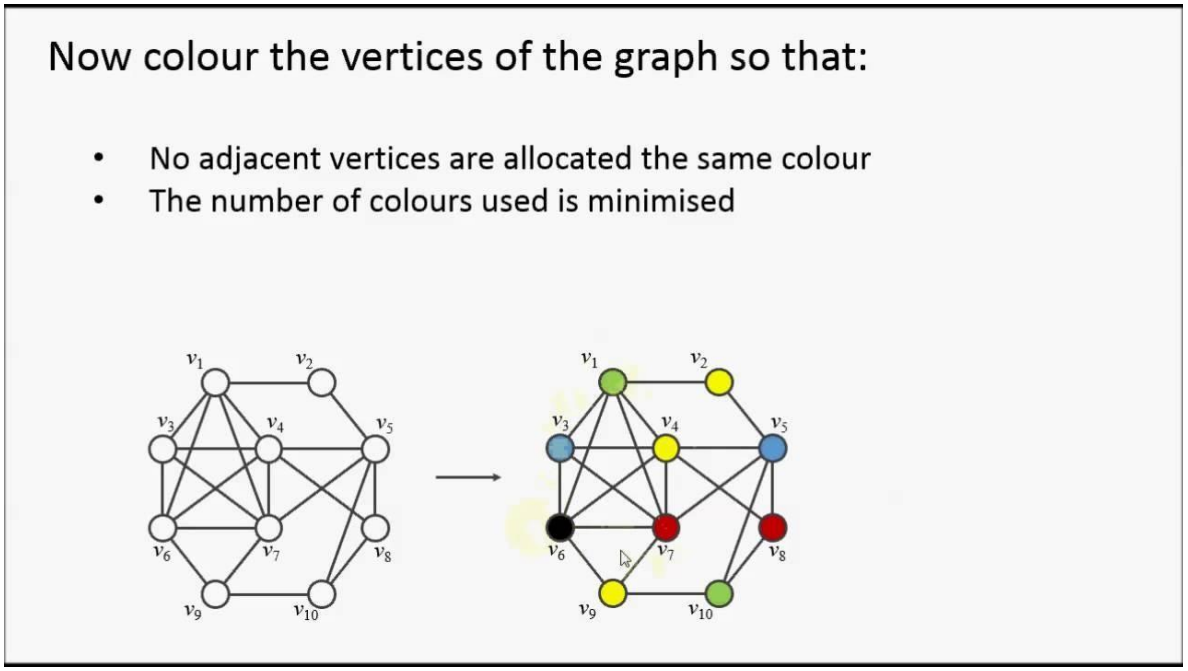


Figure 1.3 Example of a graph whose vertex is colored using graph coloring000000000

CHAPTER 2

SYSTEM SPECIFICATION

2.1 Software Requirements

Operating system	:	Windows 10
Compiler used	:	C++
Programming language	:	C ++
Editor	:	Dev C++
Graphics library	:	GL and GLU / GLUT

2.2 Hardware Requirements

Processor	:	Intel(R) Core(TM) i3-7100U CPU @ 2.40GHz 2.40 GHz
RAM size	:	8 GB
Display	:	800x600 or higher resolution display with 256 colours
Mouse	:	Standard serial mouse
Keyboard	:	Standard QWERTY keyboard
GPU	:	Intel HD Graphics 5000 or better

CHAPTER 3

WORKING OF SATELLITE

3.1 Interaction with the Program

>This program allows the user to interact using a keyboard.
The keyboard functions are listed as below:

- **"S" key starts the program and begins the process of simulation.**
- **"t/T" key starts the simulation process of transmission of signals from station to the satellite. This stage demonstrates the transmission and receiving of signals.**
- **"Q" key quits the program and exits the display screen of simulation.**
- **This completes the successful interaction and simulation of the desired process.**

CHAPTER 4

SYSTEM DESIGN AND IMPLEMENTATION

4.1 Introduction

Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development.

4.2 Initialization

Initialize the interaction with the windows. Initialize the display mode- double buffer and depth buffer. Initialize the various call back functions for drawing and redrawing, for mouse interface. Initialize the input and calculate functions for various mathematical calculations. Initialize the window position and size and create the window to display the output.

4.3 Flow of control

The flow of control in the below flow chart is with respect to the Texture Package. For any of the program flow chart is compulsory to understand the program. We consider the flow chart for the texture project in which the flow starts from start and proceeds to the main function after which it comes to the initialization of call back functions and further it proceeds to mouse and keyboard functions, input and calculation functions. Finally, it comes to quit, the end of flow chart.

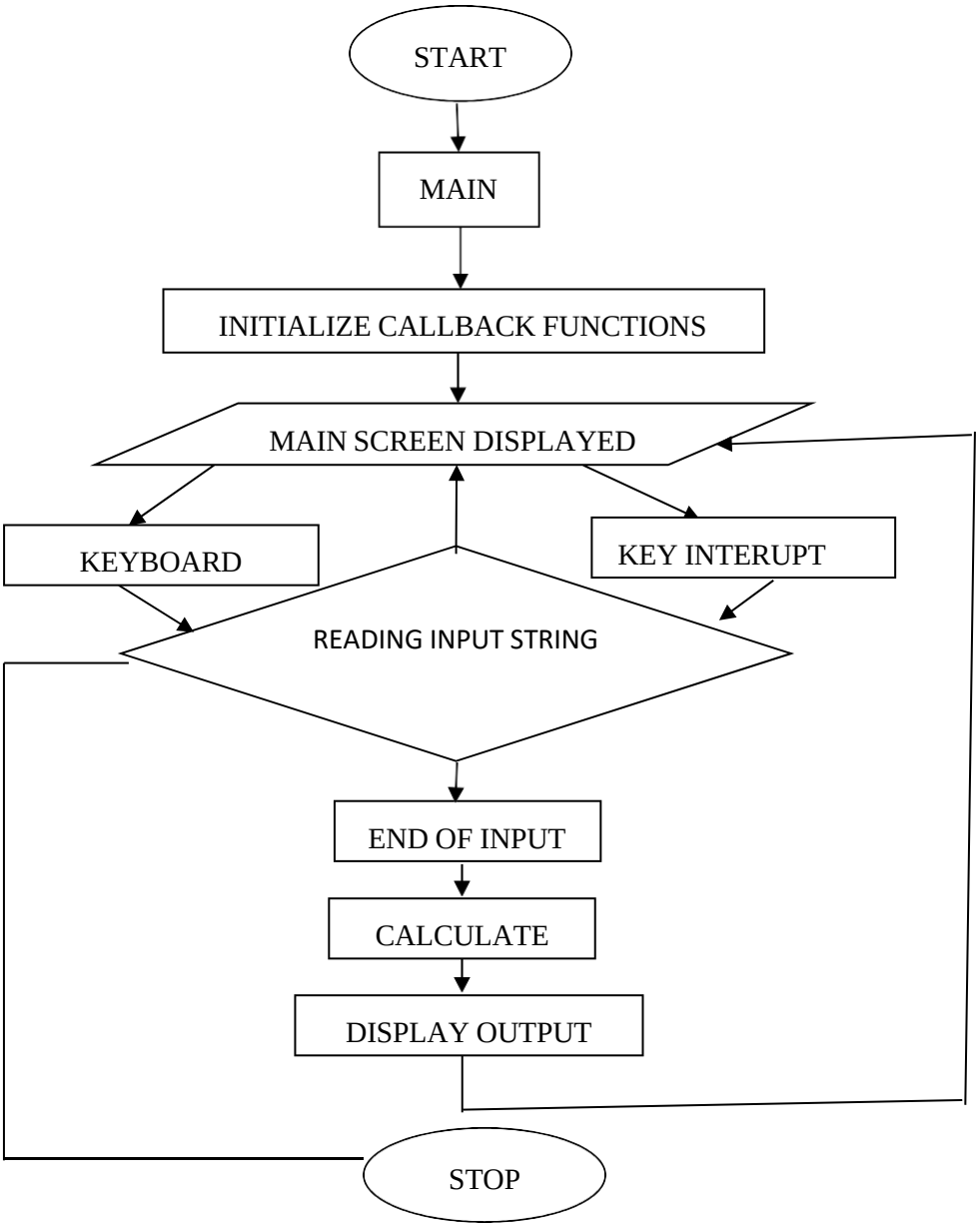


Figure 4.1 Flow Chart with respect to Texture Package

4.4 OpenGL APIs used/Built-in functions

- **glRasterPos3f()**

  Specifies the raster position for pixel operations.

- **glutBitmapCharacter()**

  Renders a bitmap charater using OpenGL from the specified array of characters, and in the specified font style.

- **glutPostRedisplay()**

  Marks the current window as needing to be redisplayed.

- **glClearColor ()**

  Specifies clear values for the color buffers.

- **glShadeModel ()**

  Select flat or smooth shading. Specifies a symbolic value representing a shading technique. Accepted values are GL_FLAT and GL_SMOOTH.

- **glEnable()**

  Enables the OpenGL capabilities, Specifies the conditions under which the pixels will be drawn.

- **glLightfv()**

  Creates a new light source with specified parameter values.

- **gluQuadricDrawStyle()**

  Specifies the draw style required for quadrics.

- **glMatrixMode ()**

  Specifies which matrix is the current matrix.

- **glLoadIdentity()**

  Pushes the identity matrix to the top of the matrix stack.

- **glutSwapBuffers()**

  Swaps the buffers of the *current window* if double buffered.

- **glViewport()**

  Sets the viewport.

- **glutInitDisplayMode ()**

  Sets the initial display mode.

- **glutInitWindowSize ()** and **glutInitWindowPosition ()**

  Set the initial window size and position respectively.

- **glutCreateWindow()**

  Creates a top level window with the window name as specified.

- **glutAddMenuEntry()**

  Adds a menu entry to the bottom of the *current menu*.

- **glutAttachMenu()**

  Attaches a mouse button for the *current window* to the identifier of the *current menu.*

- **glutDisplayFunc()**

  Sets the display callback for the *current window*.

- **glutReshapeFunc()**

  Sets the reshape callback for the *current window*.

- **glutMainLoop()**

  Enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

4.5 Pseudo Codes

Graph Coloring Algorithm

isValid(vertex, colorList, col)

Input: Vertex, colorList to check, and color, which is trying to assign.

Output: True if the color assigning is valid, otherwise false.

```
Begin
   for all vertices v of the graph, do
      if there is an edge between v and i, and col =
colorList[i], then
         return false
   done
   return true
End
```

graphColoring(colors, colorList, vertex)

Input: Most possible colors, the list for which vertices are colored with which color, and the starting vertex.

Output: True, when colors are assigned, otherwise false.

```
Begin
   if all vertices are checked, then
      return true
   for all colors col from available colors, do
      if isValid(vertex, color, col), then
         add col to the colorList for vertex
         if graphColoring(colors, colorList, vertex+1) =
true, then
            return true
         remove color for vertex
   done
   return false
End
```

4.6 Source code:

```
#include <windows.h>
#include<string.h>
#include<stdarg.h>

#include<stdio.h>
#include <glut.h>
#include  <math.h>
static double x=0.0;
static double move=-60;
static float rx[100]={0}, ry[100]={0};
//control waves
static double w1=0,w2=0,w3=0;
static bool transmit=false;
void *font;
void *currentfont;
```

```
void setFont(void *font)
{
currentfont=font;
}
void drawstring(float x,float y,float z,char *string)
{
char *c;
glRasterPos3f(x,y,z);

for(c=string;*c!='\0';c++)
{ glColor3f(0.0,1.0,1.0);
glutBitmapCharacter(currentfont,*c);
}
}
void
stroke_output(GLfloat x, GLfloat y, char *format,...)
{
va_list args;
char buffer[200], *p;
va_start(args, format);
vsprintf(buffer, format, args);
va_end(args);
glPushMatrix();
glTranslatef(-2.5, y, 0);
glScaled(0.003, 0.005, 0.005);
for (p = buffer; *p; p++)
glutStrokeCharacter(GLUT_STROKE_ROMAN, *p);
glPopMatrix();
}

void satellite(){
glRotatef(60,1,0,0);
//body
glPushMatrix();
glColor3f(0.2,0.2,0.2);
glScaled(1,0.6,1);
glTranslatef(3.0,0,0.0);
glutSolidCube(0.4);
glPopMatrix();
//Solar Panels
glPushMatrix();
glColor3f(0.3,0.3,0.3);
```

```
glTranslatef(3,0,0.0);
//glRotatef(45,1,0,0);
glScaled(3.7,0.0,1);
glutSolidCube(0.4);
glPopMatrix();

glPushMatrix();
glColor3f(0.2,0.1,0.1);
glTranslatef(3.0,0,-0.4);
glScaled(0.5,0.5,0.5);
glutSolidSphere(0.3,50,50);
glPopMatrix();
glPushMatrix();
glColor3f(0.2,0.2,0.1);
glTranslatef(3.0,0,0.4);
glScaled(0.4,0.4,0.3);
glutSolidTorus(0.3,0.2,20,20);
glPopMatrix();
}
// Second  Screen
void sat2(double ang)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glTranslatef(0.0f,0.0f,-13.0f);

glRotatef(ang,0.0f,1.0f,0.0f);
//earth
glPushMatrix();
glColor3f(0.3,0.6,1);
//glScaled(0.8,0.04,0.8);
//glTranslatef(0.0,0,0.0);
glutSolidSphere(2.0,50,50);
glPopMatrix();
satellite();
glFlush();
glutSwapBuffers();
}
void building(float x1,float y1,float z1){
//Main Structure

glPushMatrix();
```

```
glColor3f(0.5,0.5,0.5);
glTranslatef(x1,y1,z1);
glScaled(0.5,1.5,0.5);
glutSolidCube(2);
glPopMatrix();
//Dish on top
glPushMatrix();
glColor3f(1,1,0);
glTranslatef(x1,y1+1.8,z1);
glRotatef(60,1,0,0);
glScaled(0.5,1.5,0.5);
glutSolidCone(0.5,1,20,20);
glPopMatrix();
//windows
glPushMatrix();
glColor3f(0.1,0,0);
glTranslatef(x1-0.2,y1+0.7,z1);
glScaled(0.5,0.5,0.5);
//glutSolidCube(.3);
for(float j=-3;j<1.5;j+=.8)

{
for(float i=0;i<1;i+=0.8)
{
glPushMatrix();
glTranslatef(i,j,1);
glutSolidCube(0.4);
glPopMatrix();
}
}
glPopMatrix();
}
void waves(){
glPushMatrix();
glTranslatef(0,1,0);
glScaled(0.05,0.5,0.1);
glutSolidCube(0.5);
glPopMatrix();

glPushMatrix();
glRotatef(-8,0,0,1);
glTranslatef(0.01,1,0);
```

```
glScaled(0.05,0.5,0.1);
glutSolidCube(0.5);
glPopMatrix();
glPushMatrix();
glRotatef(8,0,0,1);
glTranslatef(-0.01,1,0);
glScaled(0.05,0.6,0.1);
glutSolidCube(0.5);
glPopMatrix();
}
void sat1(){
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glTranslatef(0.0f,0.0f,-13.0f);

//glRotatef(x,0.0f,1.0f,0.0f);
//Moon
glPushMatrix();
glColor3f(1,1,1);
glTranslatef(-3.8,2.8,0);
glScaled(0.5,0.5,0.1);
glutSolidSphere(0.6,50,50);
glPopMatrix();
//Earth
glPushMatrix();
glColor3f(0.2,0.2,1);
glTranslatef(0,-12,0);
//glScaled(0.8,0.04,0.8);
glutSolidSphere(10.0,50,50);
glPopMatrix();
//Building Center
glPushMatrix();
glColor3f(0,1,1);
glRotatef(10,1,0,0);
building(1.2,-1.2,3.2);

glPopMatrix();
//Building left
glPushMatrix();
glColor3f(0,1,1);
glRotatef(5,0,0,1);
building(-3.8,-1.2,0);
```

```
glPopMatrix();
//signal
glPushMatrix();
glColor3f(0,0,1);
if(transmit){
glRotatef(-25,0,0,1);
glTranslatef(-1.25,-1.6+w1,0);
}else glTranslatef(1,20,3.3);
waves();
glPopMatrix();
//Main Dish

//Tower
glPushMatrix();
glColor3f(1,1,1);
glTranslatef(-1,-2,4);
glRotatef(270,1,0,0);
glScaled(1.0,1,2.0);
glutWireCone(0.5,1,4,10);
glPopMatrix();
//Dish
glPushMatrix();
glColor3f(1,1,1);
glTranslatef(-1.08,0.2,3);
glRotatef(60,1,0,0);
glScaled(0.7,1.3,0.7);
glutSolidCone(0.4,0.5,20,20);
glPopMatrix();
//Building right
glPushMatrix();
glColor3f(0,1,1);
glRotatef(-5,0,0,1);

building(3.8,-1.2,0);
glPopMatrix();
//Saltellite
glPushMatrix();
glTranslatef(-3,3.0,0);
satellite();
glPopMatrix();
//Ack to right building
glPushMatrix();
```

```
if(transmit){
glRotatef(50,0,0,1);
glTranslatef(2.8,3.2-w2,0);
}else glTranslatef(1,20,3.3);
waves();
glPopMatrix();
//Ack to Left building

glPushMatrix();
if(transmit){
glRotatef(-50,0,0,1);
glTranslatef(-2.8,3.2-w2,0);
}else glTranslatef(1,20,3.3);
waves();
glPopMatrix();
//Ack to Center building
glPushMatrix();
if(transmit){
glRotatef(23,0,0,1);
glTranslatef(1,3.2-w3,3.3);
}
else glTranslatef(1,20,3.3);
waves();
glPopMatrix();
//stars

glPointSize(5);
for(int j=0;j<100;j++)
{
for(int i=0;i<100;i++)
{
rx[j]=rand()/500;
ry[i]=rand()/500;
glBegin(GL_POINTS);
glColor3f(0,2,2);
glVertex3f(-6+rx[j],ry[i],-5);
glEnd();
}
}
glPushMatrix();
//glScaled(1.1,2.0,0.1);
glTranslatef(0.0,0.0,-2.0);
```

```
setFont(GLUT_BITMAP_TIMES_ROMAN_24);

glColor3f(1,1,1);
drawstring(1,3.7,-1.0,"Satelitte");
setFont(GLUT_BITMAP_TIMES_ROMAN_24);
glColor3f(1,1,1);
drawstring(-4.4,.5,-1.0,"Reciever");
setFont(GLUT_BITMAP_TIMES_ROMAN_24);
glColor3f(1,1,0);
drawstring(0,-2,7,"Reciever");
setFont(GLUT_BITMAP_TIMES_ROMAN_24);
glColor3f(1,1,1);
drawstring(-1.5,-1,-1.0,"Transmitter");
setFont(GLUT_BITMAP_TIMES_ROMAN_24);
glColor3f(1,1,1);
drawstring(3.2,1,3,"Reciever");
glPopMatrix();
glFlush();
glutSwapBuffers();
}

// Third Screen
void sat3(double ang)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glTranslatef(0.0f,0.0f,-13.0f);
glRotatef(ang,0.0f,1.0f,0.0f);
//earth
glPushMatrix();
glColor3f(0.3,0.6,1);
//glScaled(0.8,0.04,0.8);
//glTranslatef(0.0,0,0.0);
glutSolidSphere(2.0,50,50);
glPopMatrix();
satellite();
glFlush();
glutSwapBuffers();
}



void e()
```

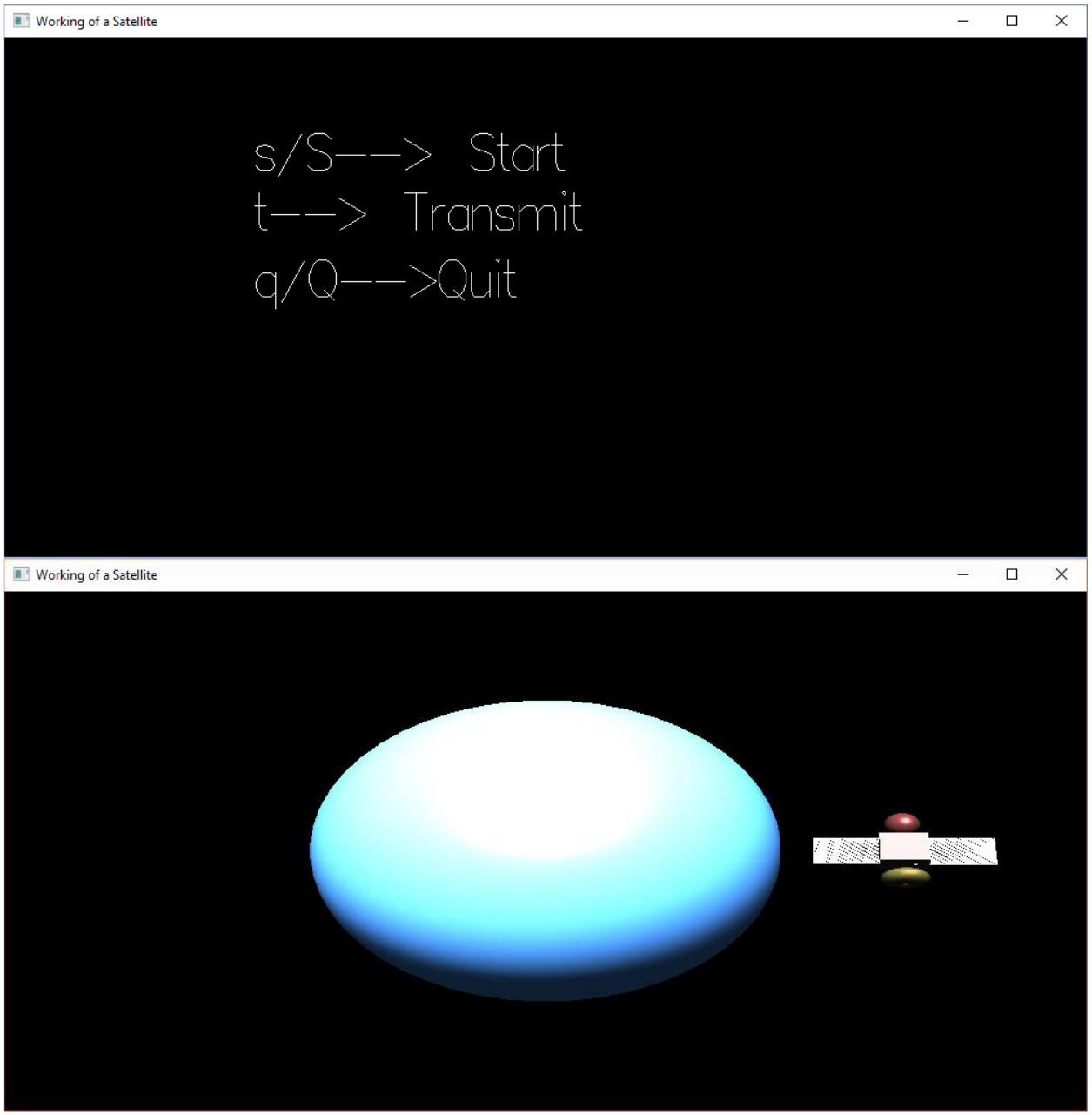
```
{
x-=0.07;
sat2(x);
}
void s()
{
x-=0.07;
sat2(x);
}
void S()
{
x += .07;glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
glutInitWindowSize(1000,480);
glutInitWindowPosition(0,0);
glutCreateWindow("Working of a Satellite");
glutDisplayFunc(display);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
glShadeModel(GL_SMOOTH);
glEnable(GL_DEPTH_TEST);
glEnable(GL_NORMALIZE);
glutKeyboardFunc(mykey);
glutCreateMenu(menu);
glutAddMenuEntry("Pyramid 's'",1);
glutAddMenuEntry("Reverse Pyramid 'S'",2);
glutAddMenuEntry("Quit 'q'",5);
glutAttachMenu(GLUT_RIGHT_BUTTON);
doInit();
glutMainLoop();

return 0;
}
```

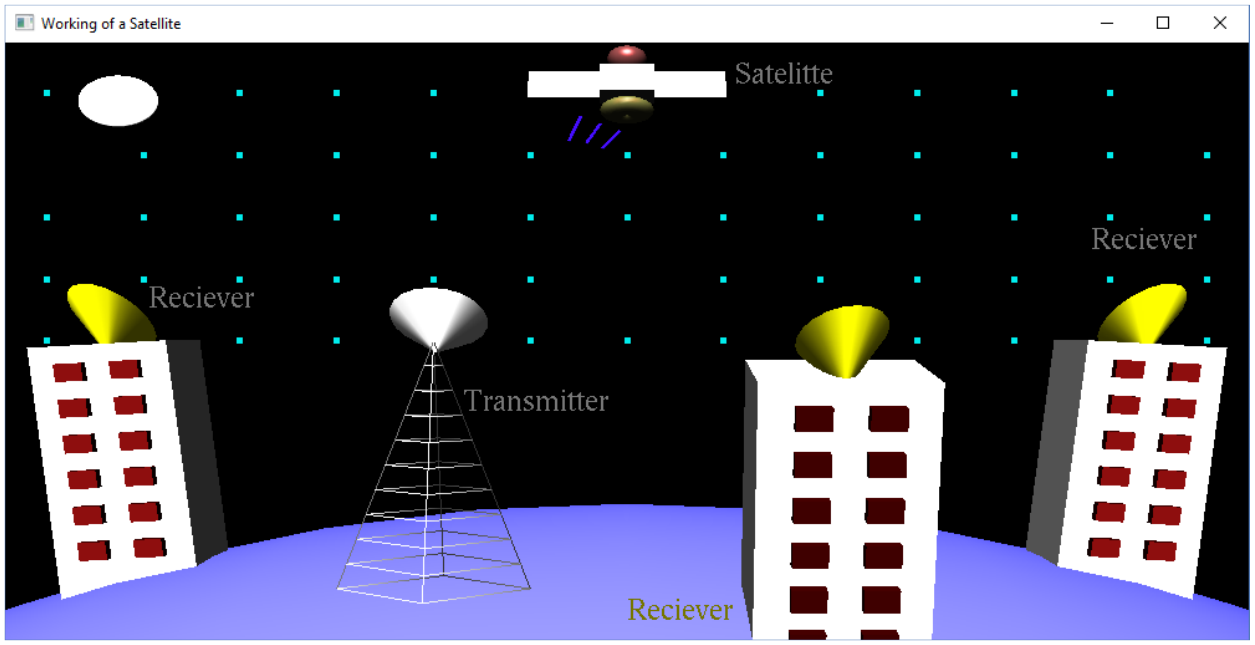
CHAPTER 5

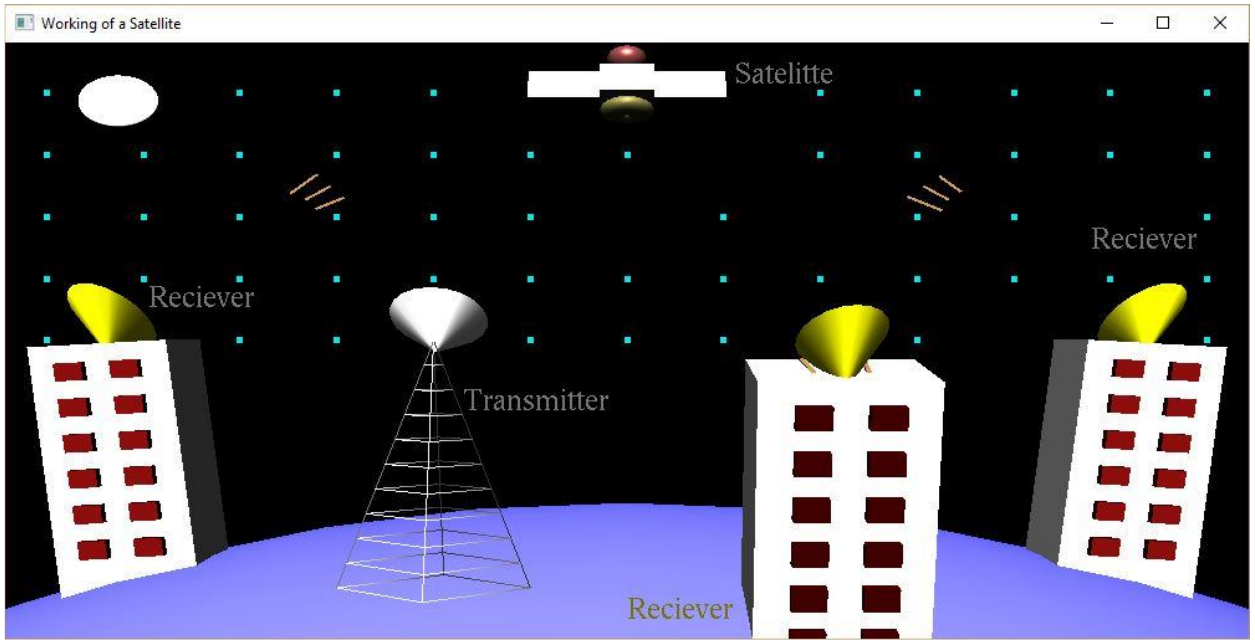
RESULTS AND DISCUSSIONS

The simulation process is thus completed successfully and the implementation of OpenGL functions is demonstrated. This stage completes the demonstration of our mini-project titled "Working of Satellite using OpenGL".
We are again grateful to our mentor Mrs. Veena Bhatt (Assistant Professor) for making us understand the whole concepts behind the OpenGL functions and not just the implementations alone.

5.1 Screenshots of the project:




Working of a Satellite
Satelitte
Reciever
Reciever
Transmitter
Reciever


Working of a Satellite
Satelitte
Reciever
Reciever
Transmitter
Reciever

Conclusion

The project “Working of a Satellite” demonstrates how signals are transmitted and received to and from a satellite.
Finally, we conclude that this program that clearly illustrates the working of a satellite using OpenGL and has been completed successfully and is ready to be demonstrated.

Thank You.

REFERENCES

Web Sources

[1] https://en.wikipedia.org/wiki/Computer_graphics#Concepts_and_principles

[2] https://en.wikipedia.org/wiki/OpenGL_Utility_Toolkit

[3] https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

[4] https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm/

Books

[5] Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version,3rd Edition, Pearson Education,2011

[6] Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL,5th edition. Pearson Education, 2008