CHAPTER 1
INTRODUCTION

Computer Graphics is concerned with the pictorial synthesis of real or imaginary objects from their computer-based models. Interactive graphics can be formally defined as methods and technologies of converting data to and from via computer.

Introduction to applications of computer graphics

The four major applications of computer graphics are as follows

- Display of information
- Design
- Simulation and animation
- User interfaces

Display of information

Information generated by architects, mechanical designers and drafts people use computer based drafting systems. Cartographers have developed maps to display celestial and geographical information. These maps were crucial to navigators. Now maps can be developed and manipulated in real time over the internet. Computer plotting packages are available that provide a variety of plotting techniques and color tools That can handle multiple large data sets. It is human's ability to recognize visual patterns that allow us to interpret the information contained in data .Medical imaging technologies like

- Computed tomography[CT]
- Magnetic resonance imaging[MRI]
- Ultrasound
- Positron emission tomography[PET]

These generate 3D data to provide meaningful information. Although the data were collected by a medical imaging system, computer graphics produced the image that shows the structural information.

Design

Professions such as engineering and architectural are concerned with design. They start with a set specification seek cost-efficient solution 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 fields including as architecture, mechanical engineering, the design of very-large-scale integrated [VLSI] circuits and creation of characters for animation.

Simulation and Animation

Once the graphics systems evolved to be capable of generating sophisticated images in real time engineers and researchers began to use them as simulators. Graphical flight simulators have provided to increase the safety and to reduce the training expenses. The use of VLSI chips has led to a generation of arcade games as sophisticated as flight simulator. 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 images with left eye and right eye which gives the effect of stereoscopic vision. The graphics to drive interactive video games make heavy use of both standard commodity computers and specialized hardware.

User Interfaces

Interaction with computers has become dominated by a visual paradigm that includes windows, icons, menus and a pointing device such as a mouse. Millions of people are internet users, they access the net through graphical network browsers such as Firefox and internet explorer.

CHAPTER 2

REQUIREMENTS SPECIFICATION

2.1 Software Requirements

- Operating System : Windows 7
- Graphics Library-glut.h
- Language Used : C++ language
- Editor :Dev-c++

2.2 Hardware Requirements

- Main processor : PENTIUM III
- Processor Speed : 800 MHz
- RAM Size : 128 MB DDR
- Keyboard : Standard qwerty serial or PS/2 keyboard
- Mouse : Standard serial or PS/2 mouse
- Compatibility : AT/T Compatible
- Cache memory : 256 KB
- Diskette drive A : 1.44 MB, 3.5 inches

CHAPTER 3

OVERVIEW

3.1 Computer Graphics

Computer Graphics is concerned with all aspects of producing pictures or images. The term computer graphics includes almost everything on computers that is not text or sound.

The term Computer Graphics has several functions:

- The various technologiesused
- To create and manipulate such pictorial data
- The imagesso produced, and the sub-field of computer sciencewhich studies methods for digitally synthesizing and manipulating visual content, see study of computer graphics

Today computers and computer-generated images touch many aspects in real world. Computer imagery is found on television, in newspapers, in weather reports, and during surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. Such graphs are used to illustrate papers, reports, theses, and other presentation material. A range of tools and facilities are available to enable users to visualize their data, and computer graphics are used in many disciplines.

3.2 OpenGL Overview

OpenGL (**Open G**raphics **L**ibrary) is a standard specification for writing applications that produce 2Dand 3D computer graphics. The data provided in an OpenGL-useable form and OpenGL will show this data on the screen

It is developed by many companies and it is free to use. OpenGL-applications can be developed 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.

GLUT is a complete API written by Mark Kilgard which allows to create windows and handle the messages. It exists for several platforms, that means that a program which

uses GLUT can be compiled on many platforms without (or at least with very few) changes in the code.

OpenGL is a state machine, that put it into various states (or modes) that then remain in effect until it is changed. For example, the current color is a state variable. The current color can be set to white, red or any color and thereafter every object is drawn with that color until the current color setting is changed. The current color is only one of many state variables that OpenGL maintains. Other control such things as the current viewing and projection transformation line and polygon stipple patterns, polygon drawing models, pixel-packing conventions, positions and characteristics of lights, and material properties of the objects being drawn.

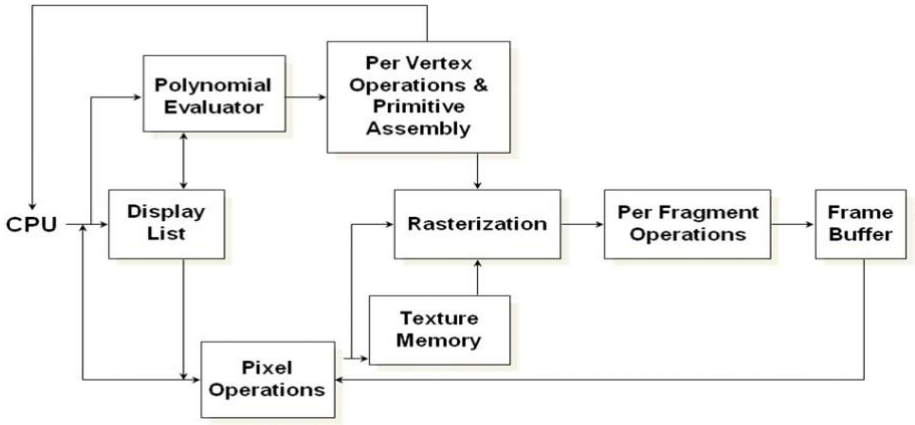

Fig 3.1: Flow of Graphical Information

This is the most important diagram, representing the flow of graphical information, as it is processed from CPU to the frame buffer. There are two pipelines of data flow. The upper pipeline is for geometric, vertex-based primitives. The lower pipeline is for pixelbased, image primitives. Texturing combines the two types of primitives together.

CHAPTER 4

ANALYSIS & DESIGN

4.1 ANALYSIS

- The simulation involves analyzing the day and night to find the rotation, simulation and scaling.
- Depending on this we can add and eliminate the trees and house displayed on the window.
- We have provided a mouse interface for the selection of various options presented in the project. The house and trees can also be altered by using the options available in the menu box.

4.2 DESIGN

- **Design Specification of day and night:** A Sun and moon can be generated. Design the corresponding menu generated on right click of the mouse. Design the functions required to do the needed tasks.
- **Design of :** The Design of transition from day to night involves transition code.The Corresponding rules of the project are implemented in a nested call of procedures initiated by the mouse function.

CHAPTER 5

INTERFACE

5.1 Library Functions Used In openGL #include<stdio.h>

The standard input/output library requires the use of a header file called stdio.h. They include command is a directive that tells the compiler to use the information in the header file called stdio.h. The initials stdio stands for standard input output, and stdio.h file contains all the instructions the compiler needs to work with disk files and send information to the output device.

#inlcude<GL/glut.h>

GL is the fundamental OpenGL library. It provides functions that are a permanent part of OpenGL. The functions start with characters 'gl'.

'glut', the GL utility tool kit supports developing and managing menus, and managing events. The functions start with characters 'glut'.

GLU, the GL Utility Library provides high-kevel routines to handle certain matrix operations, drawing of quadratic surfaces such as sphere and cylinders. The functions start with characters 'glu'.

#include<stdlib.h>

The ISO C standard introduced this header file as a place to declare certain standard library functions. These include the memory management functions that communicate with the environment (abort, exit) and others. Not all the functions of this header are supported. If a function is not there, it will be added in time.

Description of the functions main()

The execution if the program starts from main (). This is the entry point for the program.

glutInit()

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

glutInitDisplayMode()

Requests display with the properties in mode. The value of mode is determined by the logical OR of options including the colour model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE)

glutInitWindowSize()

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

glutCreateWindow()

This creates a window for the display. The string 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.

glutDisplayFunc()

It registers the display function that is executed when the window needs to be redrawn.

glutIdleFunc()

It registers the display call back function that is executed whenever there are no other events to be handled.

glutMainLoop()

It causes the program to enter an event-processing loop. It should be the last statement in main.

glClearColor()

It sets the present RGBA clear colour used when clearing the colour buffer. Variables are floating point numbers between 0.0 and 1.0.

glMatrixMode()

It specifies which matrix will be affected by subsequent transformations. Mode can be GL_MODELVIEW, GL_PROJECTION or GL_TEXTURE.

glColor()

It sets the present RGB colours. Valid types are bytes, int, float, double, unsigned byte, unsigned short and unsigned int. The maximum and minimum floating pint numbers are 1.0 and 0.0 respectively.

glutSwapBuffers()

It swaps the front and back buffers.

glFlush()

It forces any buffered OpenGL commands to execute.

5.2 Menu System

•**glutCreateMenu**glutCreateMenu creates a

new pop-up menu.

Usage:intglutCreateMenu(void (*func)(int value));func

The callback functions for the menu that is called when a menu entry from the menu is selected. The value passed to the callback is determined by the value for the selected menu entry.

Description glutCreateMenu creates a new pop-up menu and returns a unique small integer identifier. The range of allocated identifiers starts at one. The menu identifier range is separate from the window identifier range. Implicitly, the **current menu** is set to the newly created menu. This menu identifier can be used when calling glutSetMenu.

When the menu callback is called because a menu entry is selected for the menu, the **currentmenu** will be implicitly set to the menu with the selected entry before the callback is made.

- **glutAddMenuEntry**

glutAddMenuEntry adds a menu entry to the bottom of the **current menu**.

Usage:void glutAddMenuEntry(char *name, int value);

ASCII character strings to display in the menu entry. Value to be return to the menu's callback function if the menu entry is selected.

Description glutAddMenuEntry adds a menu entry to the bottom of the current menu. The string name will be displayed for the newly added menu entry. If the menu entry is selected by the user, the menu's callback will be called passing value as the callback's parameter.

- **glutAddSubMenu**

glutAddSubMenu adds a sub-menu trigger to the bottom of the **current menu**.

Usage:voidglutAddSubMenu(char *name, int menu);

ASCII character string to display in the menu item from which to cascade the sub-menu. Identifier of the menu to cascade from this sub-menu menu item.

Description glutAddSubMenu adds a sub-menu trigger to the bottom of the **current menu**. The string name will be displayed for the newly added sub-menu trigger. If the sub-menu trigger is entered, the sub-menu numbered menu will be cascaded, allowing sub-menu menu items to be selected.

5.3 Lighting

Lighting ENABLED or DISABLED?

The first - and most basic - decision is whether to enable lighting or not.

glEnable (GL_LIGHTING);...or...glDisable (GL_LIGHTING);

If it's disabled then all polygons, lines and points will be colored according to the setting of the various forms of the glColor command. Those colors will be carried forward without any change other than is imparted by texture or fog if those are also enabled with GL_LIGHTING enabled, that has to be specified more about the surface than just its color - it should be known that how shiny it is, whether it glows in the dark and whether it scatters light uniformly or in a more directional manner.

The OpenGL light model presumes that the light that reaches your eye from the polygon surface arrives by four different mechanisms:

- **AMBIENT** - light that comes from all directions equally and is scattered in all directions equally by the polygons in your scene. This isn't quite true of the real world -but it's a good first approximation for light that comes pretty much uniformly from the sky and arrives onto a surface by bouncing off so many other surfaces that it might as well be uniform.
- **DIFFUSE** - light that comes from a particular point source (like the Sun) and hits surfaces with an intensity that depends on whether they face towards the light or away from it. However, once the light radiates from the surface, it does so equally in all directions. It is diffuse lighting that best defines the shape of 3D objects.
- **SPECULAR**- as with diffuse lighting, the light comes from a point source, but with specular lighting, it is reflected more in the manner of a mirror where most of the light bounces off in a particular direction defined by the surface shape. Specular lighting is what produces the shiny highlights and helps to distinguish between flat, dull surfaces such as plaster and shiny surfaces like polished plastics and metals.

- **EMISSION** - in this case, the light is actually emitted by the polygon - equally in all directions.

5.4 Translation

The translation command glTranslate()Multiplies the current matrix by a matrix that moves (translates) an object by the given *x*, *y*, and *z* values (or moves the local coordinate system by the same amounts). glTranslate{fd}();

5.5 Scaling

The scaling command glScale()multiplies the current matrix by a matrix that stretches, shrinks, or reflects an object along the axes. Each *x*, *y*, and *z* coordinate of every point in the object is multiplied by the corresponding argument *x*, *y*, or *z*. With the local coordinate system approach, the local coordinate axes are stretched, shrunk, or reflected by the *x*, *y*, and *z* factors, and the associated object is transformed with them.

The glScale*()is the only one of the three modeling transformations that changes the apparent size of an object: scaling with values greater than 1.0 stretches an object, and using values less than 1.0 shrinks it. Scaling with a -1.0 value reflects an object across an axis.

glScale{fd }();

5.6 Rotation

The rotation command glRotate()multiplies the current matrix that rotates an object in a counterclockwise direction about the ray from the origin through the point (x,y,z). The angle parameter specifies the angle of rotation in degrees. An object that lies farther from the axis of rotation is more dramatically rotated (has a larger orbit) than an object drawn near the axis.glRotate{fd}();

CHAPTER 6

SNAPSHOTS

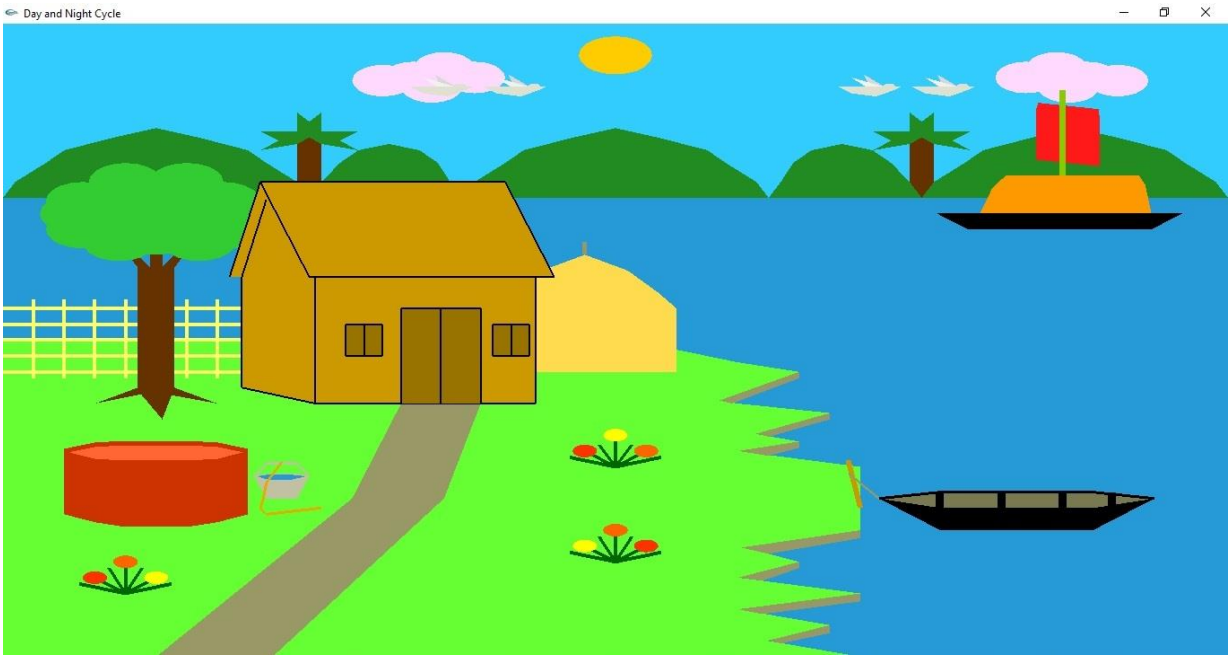

Figure 6.1 Snapshot with Day color

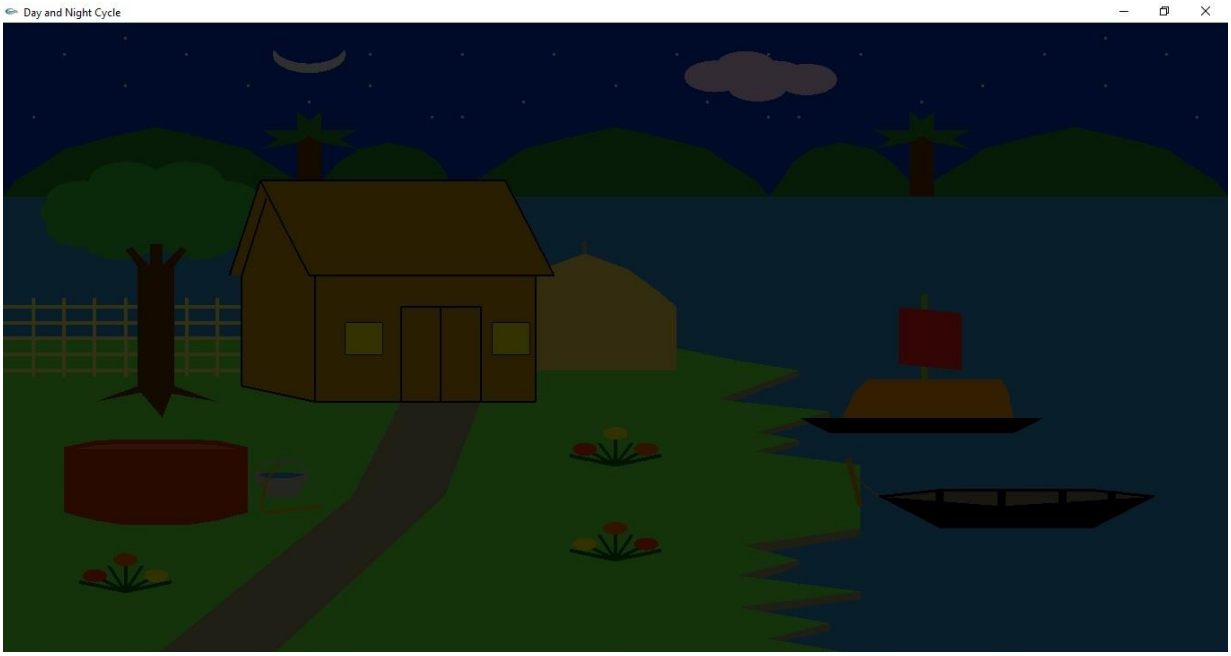

Figure.6.2 Snapshot with Night color change [done by Pressing (N)]

CHAPTER 7

CONCLUSION

We would like to conclude that our project is executed in a efficient way, which shows a beautiful view of an colored object moving in space. It also have multiple options to view it in different ways, like it can be viewed as multiple saucers moving around by adding the saucers, if there is too many saucers added we can even delete last saucer, it can also be viewed in full screen and this is how our project runs. It can be further enhanced by adding few more views and other objects in the space. We put our complete effort and gave our best it make it execute to the extinct.

FUTURE SCOPE

We have implemented the essential features of the Day and Night Cycle to our best knowledge. Even still, we would like to enhance the quality and appearance of the Day and Night Mode in the following ways:

- Enhanced appearance
- Implementation of Rain Fall

BIBLIOGRAPHY

[1] Edward Angel: Interactive Computer Graphics A Top-Down Approach With OpenGL, 5th Edition, Addison-Wisley,2008

[2] F.S.Hill,Jr: Computer Graphics Using OpenGL, 2nd Edition, Pearson Education,2001

[3] Interactive Computer Graphics – A Top-Down Approach Using OpenGL.

[4] Computer Graphics – James D Foley , Andries Van Dam , Steven K Feiner , John F Hughes , Addison – Wesley

[5] Computer Graphics – OpenGL Version – Donald Hearn and Pauline Baker , 2nd Edition , Pearson Education ,2003

APPENDIX-A

A.1 Source code

```
#include<cstdio>
#include <windows.h>
#include<math.h>
#include <vector>
#include <cstdlib>
# define PI 3.14159265358979323846
#include <GL/gl.h>
#include <GL/glut.h>
#include<MMSystem.h>
void PointLight(const float x, const float y, const float z,  const float amb, const float diff,
const float spec);
void PointLight(const float x, const float y, const float z, const float amb, const float diff,
const float spec)
{
  glEnable(GL_LIGHTING);
  GLfloat light_ambient[] = { amb,amb,amb, 1.0 };
  GLfloat light_position[] = {-0.9,.9,0, 0.0 };
  glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  glEnable(GL_LIGHT0); //enable the light after setting the properties
}
GLfloat position22 = 0.0f;
GLfloat speed22 = 0.007f;
void birdd(int value) {
    if(position22 > 1.0)
        position22 =-1.0f;
    position22 += speed22;
     glutPostRedisplay();
     glutTimerFunc(100, birdd, 0);
}
GLfloat position4 = 0.0f;
GLfloat speed4 =-0.01f;
void sunn(int value)
{
   if(position4 > 1.0)
        position4 = 0.0f;

    position4 += speed4;

     glutPostRedisplay();
```

```
    glutTimerFunc(100, sunn, 0);
}

GLfloat position2 = 0.0f;
GLfloat speed2 = 0.004f;
void cloud(int value) {
   if(position2 > 1.0)
      position2 =-1.0f;
   position2 += speed2;
    glutPostRedisplay();
    glutTimerFunc(100, cloud, 0);
}
GLfloat position1 = 1.0f;
GLfloat speed1 =-0.005f;
void boat(int value)
{
  if(position1 <- 1.0)
      position1 = 1.0f;

   position1 += speed1;

    glutPostRedisplay();



    glutTimerFunc(100, boat, 0);
}
void cloud1()
{
   int i;

    GLfloat x=.5f; GLfloat y=.86f; GLfloat radius =.05f;
    int triangleAmount = 20;
    GLfloat twicePi = 2.0f * PI;

    glBegin(GL_TRIANGLE_FAN);
       glColor3ub(255, 217, 255);
            glVertex2f(x, y); // center of circle
            for(i = 0; i <= triangleAmount;i++) {
                    glVertex2f(
                    x + (radius * cos(i *  twicePi / triangleAmount)),
                       y + (radius * sin(i * twicePi / triangleAmount))
                    );
            }
    glEnd();
```

```
GLfloat a=.55f; GLfloat b=.83f;

    glBegin(GL_TRIANGLE_FAN);
       glColor3ub(255, 217, 255);
            glVertex2f(a, b); // center of circle
            for(i = 0; i <= triangleAmount;i++) {
                    glVertex2f(
                    a + (radius * cos(i *  twicePi / triangleAmount)),
                       b + (radius * sin(i * twicePi / triangleAmount))
                    );
            }
    glEnd();

    GLfloat c=.45f; GLfloat d=.83f;

    glBegin(GL_TRIANGLE_FAN);
       glColor3ub(255, 217, 255);
            glVertex2f(c, d); // center of circle
            for(i = 0; i <= triangleAmount;i++) {
                    glVertex2f(
                    c + (radius * cos(i *  twicePi / triangleAmount)),
                       d + (radius * sin(i * twicePi / triangleAmount))
                    );
            }
    glEnd();

    GLfloat e=.52f; GLfloat f=.8f;

    glBegin(GL_TRIANGLE_FAN);
       glColor3ub(255, 217, 255);
            glVertex2f(e, f); // center of circle
            for(i = 0; i <= triangleAmount;i++) {
                    glVertex2f(
                    e + (radius * cos(i *  twicePi / triangleAmount)),
                       f+ (radius * sin(i * twicePi / triangleAmount))
                    );
            }
    glEnd();

    GLfloat g=.6f; GLfloat h=.82f;

    glBegin(GL_TRIANGLE_FAN);
       glColor3ub(255, 217, 255);
            glVertex2f(g, h); // center of circle
            for(i = 0; i <= triangleAmount;i++) {
```

```
                glVertex2f(
                g + (radius * cos(i *  twicePi / triangleAmount)),
                    h+ (radius * sin(i * twicePi / triangleAmount))
                );
            }
    glEnd();


}

void cloud2()
{
  // glLoadIdentity();
   int i;

    GLfloat x=-.5f; GLfloat y=.86f; GLfloat radius =.05f;
    int triangleAmount = 20;
    GLfloat twicePi = 2.0f * PI;

    glBegin(GL_TRIANGLE_FAN);
       glColor3ub(255, 217, 255);
            glVertex2f(x, y); // center of circle
            for(i = 0; i <= triangleAmount;i++) {
                glVertex2f(
                x + (radius * cos(i *  twicePi / triangleAmount)),
                    y + (radius * sin(i * twicePi / triangleAmount))
                );
            }
    glEnd();

GLfloat a=-.55f; GLfloat b=.83f;

    glBegin(GL_TRIANGLE_FAN);
       glColor3ub(255, 217, 255);
            glVertex2f(a, b); // center of circle
            for(i = 0; i <= triangleAmount;i++) {
                glVertex2f(
                a + (radius * cos(i *  twicePi / triangleAmount)),
                    b + (radius * sin(i * twicePi / triangleAmount))
                );
            }
    glEnd();

    GLfloat c=-.45f; GLfloat d=.83f;

    glBegin(GL_TRIANGLE_FAN);
```

```
        glColor3ub(255, 217, 255);
            glVertex2f(c, d); // center of circle
            for(i = 0; i <= triangleAmount;i++) {
                glVertex2f(
                c + (radius * cos(i *  twicePi / triangleAmount)),
                    d + (radius * sin(i * twicePi / triangleAmount))
                );
            }
    glEnd();

    GLfloat e=-.52f; GLfloat f=.8f;

    glBegin(GL_TRIANGLE_FAN);
        glColor3ub(255, 217, 255);
            glVertex2f(e, f); // center of circle
            for(i = 0; i <= triangleAmount;i++) {
                glVertex2f(
                e + (radius * cos(i *  twicePi / triangleAmount)),
                    f+ (radius * sin(i * twicePi / triangleAmount))
                );
            }
    glEnd();

    GLfloat g=-.6f; GLfloat h=.82f;

    glBegin(GL_TRIANGLE_FAN);
        glColor3ub(255, 217, 255);
            glVertex2f(g, h); // center of circle
            for(i = 0; i <= triangleAmount;i++) {
                glVertex2f(
                g + (radius * cos(i *  twicePi / triangleAmount)),
                    h+ (radius * sin(i * twicePi / triangleAmount))
                );
            }
    glEnd();
}
```

A.2 Personal Details

NAME: KIRAN GOWDA . S

USN: 1DT17CS406

SEMESTER AND SECTION: 6TH SEM, A SEC

DEPARTMENT: COMPUTER SCIENCE AND ENGINEERING

EMAIL ID: kirangowdas2015@gmail.com

NAME: SUJAN KIRAN . J

USN: 1DT17CS416

SEMESTER AND SECTION: 6TH SEM, A SEC

DEPARTMENT: COMPUTER SCIENCE AND ENGINEERING

EMAIL ID: sujankiran14@gmail.com