Introduction

1.1 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 sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content, see study of computer graphics

Computer graphics is widespread today. The Computer imagery is found on television, in newspapers, for example in weather reports, or for example 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: 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".

Computer graphics is an art of drawing pictures, lines, charts, etc using computers with the help of programming. Computer graphics is made up of number of pixels. Pixel is the smallest graphical picture or unit represented on the computer screen.

1.2 Application of Computer Graphics

Paint programs: Allow you to create rough freehand drawings. The images are stored as bit maps and can easily be edited. It is a graphics program that enables you to draw pictures on the display screen which is represented as bit maps (bit-mapped graphics). In contrast, draw programs use vector graphics (object-oriented images), which scale better.

Most paint programs provide the tools shown below in the form of icons. By selecting an icon, you can perform functions associated with the tool.In addition to these tools, paint programs also provide easy ways to draw common shapes such as straight lines, rectangles, circles, and ovals.

Sophisticated paint applications are often called image editing programs. These applications support many of the features of draw programs, such as the ability to work with objects. Each object, however, is represented as a bit map rather than as a vector image.

Illustration/design programs: Supports more advanced features than paint programs, particularly for drawing curved lines. The images are usually stored in vector-based formats. Illustration/design programs are often called draw programs. Presentation graphics software: Lets you create bar charts, pie charts, graphics, and other types of images for slide shows and reports. The charts can be based on data imported from spreadsheet applications.

A type of business software that enables users to create highly stylized images for slide shows and reports. The software includes functions for creating various types of charts and graphs and for inserting text in a variety of fonts. Most systems enable you to import data from a spreadsheet application to create the charts and graphs. Presentation graphics is often called business graphics.

Animation software: Enables you to chain and sequence a series of images to simulate movement. Each image is like a frame in a movie. It can be defined as a simulation of movement created by displaying a series of pictures, or frames. A cartoon on television is one example of animation. Animation on computers is one of the chief ingredients of multimedia presentations. There are many software applications that enable you to create animations that you can display on a computer monitor.

There is a difference between animation and video. Whereas video takes continuous motion and breaks it up into discrete frames, animation starts with independent pictures and puts them together to form the illusion of continuous motion.

CAD software: Enables architects and engineers to draft designs. It is the acronym for computer-aided design. A CAD system is a combination of hardware and software that enables engineers and architects to design everything from furniture to airplanes. In addition to the software, CAD systems require a high-quality graphics monitor; a mouse, light pen, or digitizing tablet for drawing; and a special printer or plotter for printing design specifications.

CAD systems allow an engineer to view a design from any angle with the push of a button and to zoom in or out for close-ups and long-distance views. In addition, the computer keeps track of design dependencies so that when the engineer changes one value, all other values that depend on it are automatically changed accordingly. Until the middle 1980s, all CAD systems were specially constructed computers. Now, you can buy CAD software that runs on general-purpose workstations and personal computers.

Desktop publishing: Provides a full set of word-processing features as well as fine control over placement of text and graphics, so that you can create newsletters, advertisements, books, and other types of documents. It means by using a personal computer or workstation high-quality printed documents can be produced. A desktop publishing system allows you to use different typefaces, specify various margins and justifications, and embed illustrations and graphs directly into the text. The most powerful desktop publishing systems enable you to create illustrations; while less powerful systems let you insert illustrations created by other programs.

As word-processing programs become more and more powerful, the line separating such programs from desktop publishing systems is becoming blurred. In general, though, desktop publishing applications give you more control over typographical characteristics, such as kerning, and provide more support for full-color output.

A particularly important feature of desktop publishing systems is that they enable you to see on the display screen exactly how the document will appear when printed. Systems that support this feature are called WYSIWYGs (what you see is what you get). Until recently, hardware costs made desktop publishing systems impractical for most uses. But as the prices of personal computers and printers have fallen, desktop publishing systems have become increasingly popular for producing newsletters, brochures, books, and other documents that formerly required a typesetter.

Once you have produced a document with a desktop publishing system, you can output it directly to a printer or you can produce a PostScript file which you can then take to a service bureau. The service bureau has special machines that convert the PostScript file to film, which can then be used to make plates for offset printing. Offset printing produces higher-quality documents, especially if color is used, but is generally more expensive than laser printing.

In general, applications that support graphics require a powerful CPU and a large amount of memory. Many graphics applications—for example, computer animation systems—require more computing power than is available on personal computers and will run only on powerful workstations or specially designed graphics computers. This is true of all three-dimensional computer graphics applications.

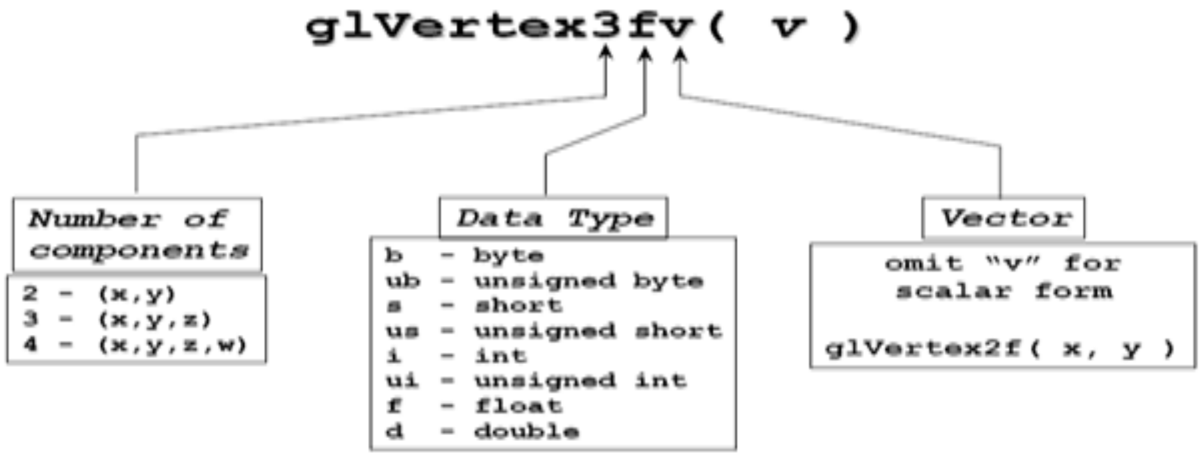
In addition to the CPU and memory, graphics software requires a graphics monitor and support for one of the many graphics standards. Most PC programs, for instance, require VGA graphics. If your computer does not have built-in support for a specific graphics system, you can insert a video adapter card.

The quality of most graphics devices is determined by their resolution—how many pixels per square inch they can represent—and their color capabilities.

1.3 OpenGL

- OpenGL is an API (Application Programming Interface) which provides the actual drawing tool through a collection of functions that are called within an application.
- It suits well for use in computer graphics because of its device independence or portability. It offers a rich and highly usable API for 2D, 3D graphics and image manipulation.
- Computer graphics is concerned with all aspects of producing pictures or images using a computer. OpenGL is a graphics software system which has become a widely accepted standard for developing graphics applications. it is easy to learn and it possesses most of the characteristics of other popular graphics system.
- OpenGL is a software interface to graphics hardware. OpenGL is a library of functions for fast 3d rendering. The openGL utility library (GLU) provides many modeling features. It is a standard part of every openGL implementation. It is ideal for 3d visualization but offers little support for creating application user interfaces.

  OpenGL command Format



Basic OpenGL Operation: The figure1.2 shown below gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

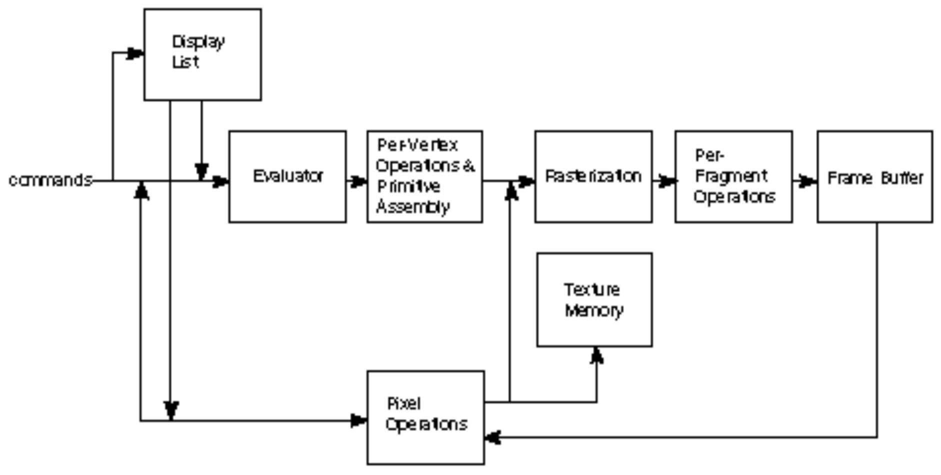


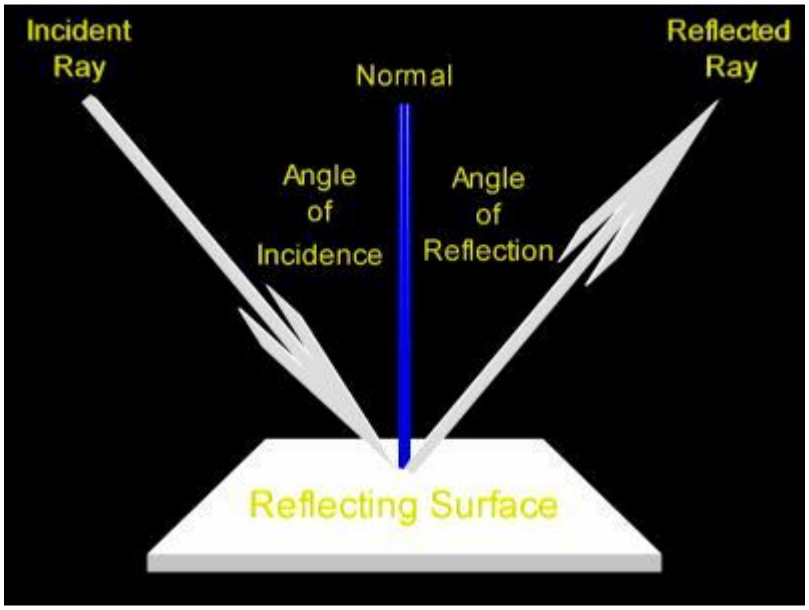
Fig 1.2: OPENGL BLOCK DIAGRAM

- The evaluator stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values.
- During the next stage, per-vertex operations and primitive assembly, OpenGL processes geometric primitives—points, line segments, and polygons, all of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for the next stage.
- Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each fragment so produced is fed into the last stage, per-fragment operations, which perform the final operations on the data before it's stored as pixels in the frame buffer. These operations include conditional updates to the frame buffer based on incoming and previously stored z-values (for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

- Input data can be in the form of pixels rather than vertices. Such data, which might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the pixel operations stage.
- The result of this stage is either stored as texture memory, for use in the rasterization stage, or rasterized and the resulting fragments merged into the frame buffer just as if they were generated from geometric data.

1.4 Statement of the given problem

Depicting reflection using ray diagrams along with the related concepts using three types of mirrors.



1.5 Objective of the project Motivation

The objective of the project is to show properties of reflection on three different mirrors.

System Specification

2.1 Hardware Specifications:

- Pentium II and above.
- 128 MB RAM and above.
- 20 GB HDD.

2.2 Software Specifications:

- Compiler: Dev C++, Microsoft Visual Studio 2006 (Visual C++)
- Operating System: Windows XP, Windows Vista, Windows 7

Analysis

- **Reflection** is the change in direction of a wavefront at an interface between two different media so that the wavefront returns into the medium from which it originated.

- High Level Description : The user can learn the basics of reflection in the three types of mirrors. They are – plane mirror, concave mirror and convex mirror.

- High level functional requirements :
  1) The arrows with different colours show the incident ray,reflected ray,object and image.
  2) There are two menus in the starting showing the concept and the ray diagrams.

- Non Functional Requirements:
  1) The speed of learning the concept of reflection depends on the user's capabitlity.
  2) It has keyboard interface and and mouse interface.
  3) Delay is present to show the occurance of incident ray,reflected ray and image formation.

Implementation

4.1 InBuilt Functions

- The openGL provides very powerful functions which relieve the programmes by allowing them to concentrate on their job rather than focusing on how to implement various operations.

The Built-In Fuctions used are:

- void **glMatrixMode**(GLenum mode):

Specifies whether the model view, projection or texture matrix will be modified using the arguments GL_MODEL_VIEW, GL_PROJECTION or GL_TEXTURE for mode. Subsequent transformation commands affects the specified matrix.

- void **glClear(**GLbitfield mask):

Clears the specified buffers to their current clearing values.

- void **glFlush**(void):

Forces previously issued openGL commands to begin execusion, thus guaranteeing that they complete in finite time. The situations where glFlush() is useful are software renderers that build image in system memory and don't want to constantly update the screen.

- void **glutMainLoop**(void):

Enters the GLUT processing loop, never to return. Registered callback functions will be called when the corressponding eventsin instigate them.

- void **glutPostRedisplay**():

Marks the normal plane of current window as needing to be redisplayed. After the call, the next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback.

- void **glutIdleFunc**(void (*func)(void)):

Specifies the function func to be executed if no other events are pending. If NULL(0) is passed in, execusion of func is disabled.

- void**glutDisplayFunc**(void (*func)(void)):

Sets the display callback for the *current window*. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called.

- void**glutKeyboardFunc**(void (*func)(int key, int x, int y)):

Sets the special keyboard callback for the *current window*. The special keyboard callback is triggered when keyboard function or directional keys are pressed. The key callback parameter is a GLUT_KEY_* constant for the special key pressed. The x and y callback parameters indicate the mouse in window relative coordinates when the key was pressed.

- void **glutMouseFunc**(void (*func)(int button, int state,
  int x, int y));

  sets the mouse callback for the *current window*. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback. The button parameter is one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON.

- void **glutBitmapCharacter**(void *font, int character);

GLUT_BITMAP_HELVETICA_10

A 10-point proportional spaced Helvetica font. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named:

-adobe-helvetica-medium-r-normal--10-100-75-75-p-56-iso8859-1

GLUT_BITMAP_TIMES_ROMAN_10

A 10-point proportional spaced Times Roman font. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named:

-adobe-times-medium-r-normal--10-100-75-75-p-54-iso8859-1

4.2Userbuilt Functions

- void setFont(void *font)
  sets the font.
- void drawstring(float x,float y,float z,char *string)
  draws the string using the built in glut**B**itmapCharacter function.
- void draw(GLfloat x1,GLfloat x2)
  draws the polygon for display menus.
- void text(GLfloat x)
  draws the text in menu screen.
- void delay()
  to give the delays in the program.
- void shading(GLfloat x,GLfloat y)
  draws the shades of the behind mirror.
- void drawpixel(GLfloat x,GLfloat y)
  draws the pixels of the concave mirror.
- void plotpixel(GLfloat h,GLfloat k,GLfloat x,GLfloat y)
  plots the pixel for concave mirror.
- void circle(GLfloat h,GLfloat k,GLfloat r)
  draws the circle of concave mirror.

- void drawpixel1(GLfloat x,GLfloat y)
  draws the pixels of the convex mirror.
- void plotpixel1(GLfloat h,GLfloat k,GLfloat x,GLfloat y)
  plots the pixel for convex mirror.
- void circle1(GLfloat h,GLfloat k,GLfloat r)
  draws the circle of convex mirror.
- void dec()
  initial declarations on the screen.
- void planemirror1()
  shows the plane mirror concept.
- void planemirror()
  shows the plane mirror ray diagram.
- void concavemirror1()
  shows the concave mirror concept.
- void concavemirror()
  shows the concave mirror ray diagram.

- void convexmirror1()
  shows the convex mirror concept.
- void convexmirror()
  shows the convex mirror ray diagram.
- void mainmenu()
  shows the initial menu.
- void key(char *s)
  text for keyboard interaction.
- void concept()
  menu for the concept.
- void mykeyboard(unsigned char key,int x,int y)
  for keyboard interaction.
- void myMouse(int btn, int state, int x, int y)
  for mouse interaction.

4.3 SOURCE CODE

```
| #include<GL/glut.h>
#include<stdio.h>
#include<stdlib.h>
#define   drawOneLine(x1,y1,x2,y2) glBegin(GL_LINES); \
   glVertex2f ((x1),(y1)); glVertex2f ((x2),(y2)); glEnd();

void *currentfont;                                        //output char
GLint i,j,c=0;;

void setFont(void *font)
{
        currentfont=font;
}



void drawstring(float x,float y,float z,char *string)                //string display
{
            char *c;
        glRasterPos3f(x,y,z);

        for(c=string;*c!='\0';c++)
        {       glColor3f(0.0,0.0,0.0);
                glutBitmapCharacter(currentfont,*c);
        }
}

void draw(GLfloat x1,GLfloat x2) // TO DRAW POLYGON FOR DISPLAY MENUS
```

```
{

        glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);  // plane mirror
        glColor3f(0.7f,0.2f,0.2f);
   glVertex2i(x2,425);
        glVertex2i(x1,425);
        glVertex2i(x1,395);
   glVertex2i(x2,395);
        glEnd();
        glFlush();

        glBegin(GL_POLYGON);  // concave mirror
        glColor3f(0.7f,0.2f,0.2f);
   glVertex2i(x2,370);
        glVertex2i(x1,370);
        glVertex2i(x1,340);
   glVertex2i(x2,340);
        glEnd();
        glFlush();

        glBegin(GL_POLYGON);  // convex mirror
        glColor3f(0.7f,0.2f,0.2f);
   glVertex2i(x2,315);
        glVertex2i(x1,315);
        glVertex2i(x1,285);
   glVertex2i(x2,285);
        glEnd();
        glFlush();

        glBegin(GL_POLYGON);  // exit
        glColor3f(0.7f,0.2f,0.2f);
    glVertex2i(x2,150);
        glVertex2i(x1,150);
        glVertex2i(x1,120);
    glVertex2i(x2,120);
        glEnd();
        glFlush();

}

void text(GLfloat x)
        // to draw the text in menu screen
{



        setFont(GLUT_BITMAP_HELVETICA_18);
        glColor3f(1.0,1.0,1.0);
        drawstring(x,405.0,1.0,"PLANE MIRROR");

        glColor3f(1.0,1.0,1.0);
        drawstring(x,350.0,1.0," CONCAVE MIRROR");

        glColor3f(1.0,1.0,1.0);
        drawstring(x,295.0,1.0,"CONVEX MIRROR");


        glColor3f(1.0,1.0,1.0);
        drawstring(x,130.0,1.0,"   MAIN MENU");
        glFlush();
}
```

```
void myInit()
{
   glClearColor(0.0,0.0,0.0,0.0);
         glColor3f(0.0,0.0,0.0);
         glPointSize(5.0);
         gluOrtho2D(0.0,500.0,0.0,500.0);
         glMatrixMode(GL_PROJECTION);
         glLoadIdentity();
         setFont(GLUT_BITMAP_HELVETICA_18);
         glShadeModel (GL_FLAT);
}

void delay()
{
 j=28000;
         while(j!=0)
         {
                  j--;
                  i=28000;
                  while(i!=0)
                  {
                           i--;

                  }
         }
}
void shading(GLfloat x,GLfloat y)                   //drawing shades of mirror
{
         glBegin(GL_LINES);
```

```
        glVertex2f(x,y);
        glVertex2f(x+7.0,y+2.0);
        glEnd();
        glFlush();
}

void drawpixel(GLfloat x,GLfloat y)
{
        glColor3f(0.5,0.5,0.5);
        glBegin(GL_POINTS);
        glVertex2f(x,y);
        glEnd();
        c++;
        if(c%5==0)
                shading(x,y);
}
void plotpixel(GLfloat h,GLfloat k,GLfloat x,GLfloat y)      // for concave mirror
{
        drawpixel(y+h,x+k);
        drawpixel(y+h,-x+k);
}
void circle(GLfloat h,GLfloat k,GLfloat r)                   //for concave mirror
{
        GLfloat d=1-r,x=0,y=r;
        while(y>x)
        {
                plotpixel(h,k,x,y);
                if(d<0)
                        d+=2*x+3;
                else
                {
                        d+=2*(x-y)+5;
                        --y;
                }
                ++x;
        }
        plotpixel(h,k,x,y);
        glFlush();
}

void plotpixel1(GLfloat h,GLfloat k,GLfloat x,GLfloat y)          //for convex mirror
{
        drawpixel(-y+h,x+k);
        drawpixel(-y+h,-x+k);
}
void circle1(GLfloat h,GLfloat k,GLfloat r)                       //for convex mirror
{
        GLfloat d=1-r,x=0,y=r;
        while(y>x)
        {
                plotpixel1(h,k,x,y);
                if(d<0)
                        d+=2*x+3;
                else
                {
                        d+=2*(x-y)+5;
                        --y;
                }
                ++x;
        }
        plotpixel1(h,k,x,y);
        glFlush();
```

```
}

void dec()
                        //initial declaration
{

        glClear(GL_COLOR_BUFFER_BIT);

        glColor3f(1.0,1.0,0.0);
        glBegin(GL_LINES);
        glVertex2f(350,480);
        glVertex2f(370,480);
        glVertex2f(365,485);
        glVertex2f(370,480);
        glVertex2f(365,475);
        glVertex2f(370,480);
        glEnd();

        glColor3f(0.0,1.0,0.0);
        glBegin(GL_LINES);
        glVertex2f(350,470);
        glVertex2f(370,470);
        glVertex2f(365,475);
        glVertex2f(370,470);
        glVertex2f(365,465);
        glVertex2f(370,470);
        glEnd();

        glColor3f(1.0,0.0,0.0);
        glBegin(GL_LINES);
        glVertex2f(360,460);
```

```
glVertex2f(360,440);
glVertex2f(360,460);
glVertex2f(355,455);
glVertex2f(365,455);
glVertex2f(360,460);
glEnd();

glColor3f(1.0,0.0,1.0);
glBegin(GL_LINES);
glVertex2f(360,430);
glVertex2f(360,410);
glVertex2f(360,430);
glVertex2f(355,425);
glVertex2f(365,425);
glVertex2f(360,430);
glEnd();

glColor3f(0.0,0.0,1.0);
glBegin(GL_LINES);
glVertex2f(100,300);
glVertex2f(400,300);
glEnd();
glColor3f(1.0,1.0,1.0);
glBegin(GL_POINTS);
glVertex2f(250.0,300.0);
glEnd();

setFont(GLUT_BITMAP_HELVETICA_18);
glColor3f(1.0,1.0,1.0);
drawstring(240.0,290.0,1.0,"P");
glColor3f(1.0,1.0,1.0);
```

```
            drawstring(380.0,477.0,1.0,"Incident ray");

            glColor3f(1.0,1.0,1.0);
            drawstring(378.0,467.0,1.0," Reflected ray");

            glColor3f(1.0,1.0,1.0);
            drawstring(380.0,450.0,1.0,"Object");



            glColor3f(1.0,1.0,1.0);
            drawstring(375.0,420.0,1.0,"  Image");

            glFlush();
    }
    void planemirror1()
                    //plane mirror concept
    {
            dec();
            setFont(GLUT_BITMAP_HELVETICA_18);
            glColor3f(0.0,1.0,1.0);
            drawstring(200.0,455.0,1.0,"P L A N E  M I R R O R");
            glFlush();
            glColor3f(0.0,1.0,1.0);
            glBegin(GL_LINES);
            glVertex2f(195.0,450.0);
            glVertex2f(300.0,450.0);
            glEnd();
            glColor3f(0.5,0.5,0.5);
            glBegin(GL_LINES);
            glVertex2f(250,400);
            glVertex2f(250,200);
```

```
        glEnd();
        GLfloat l;
        for(l=205.0;l<400.0;l+=5.0)
                shading(250.0,l);

        glColor3f(1.0,1.0,0.0);
        //incident ray
    glBegin(GL_LINES);
        glVertex2f(100,350);
        glVertex2f(250,350);
    glVertex2f(100,250);
        glVertex2f(250,250);

        glVertex2f(245,355);
        glVertex2f(250,350);
    glVertex2f(245,345);
        glVertex2f(250,350);

        glVertex2f(245,255);
        glVertex2f(250,250);
    glVertex2f(245,245);
        glVertex2f(250,250);
        glEnd();
        glFlush();
        delay();

        glColor3f(0.0,0.0,0.0);
        //clear traces
        glBegin(GL_LINES);
        glVertex2f(245,355);
        glVertex2f(250,350);
```

```
glVertex2f(245,345);
        glVertex2f(250,350);

        glVertex2f(245,255);
        glVertex2f(250,250);
glVertex2f(245,245);
        glVertex2f(250,250);
        glEnd();

        glColor3f(0.0,1.0,0.0);
        //reflected ray
glBegin(GL_LINES);
        glVertex2f(100,350);
        glVertex2f(250,350);
glVertex2f(100,250);
        glVertex2f(250,250);

        glVertex2f(230,350);
        glVertex2f(235,355);
        glVertex2f(230,350);
        glVertex2f(235,345);

        glVertex2f(230,250);
        glVertex2f(235,255);
        glVertex2f(230,250);
        glVertex2f(235,245);
        glEnd();
        glFlush();
        delay();
```

```
            glColor3f(1.0,1.0,0.0);
            //image
    glBegin(GL_LINES);
            glVertex2f(200,325);
            glVertex2f(250,300);
    glVertex2f(250,300);
            glVertex2f(245,305);
            glVertex2f(250,300);
            glVertex2f(244,301);
            glEnd();
            glFlush();
            delay();

            glColor3f(0.0,1.0,0.0);
    glBegin(GL_LINES);
            glVertex2f(200,275);
            glVertex2f(250,300);
            glVertex2f(200,275);
            glVertex2f(203,280);
            glVertex2f(200,275);
            glVertex2f(205,275);
            glEnd();
            glFlush();
}
void planemirror()
            //plane mirror ray diagram
{
            dec();
            setFont(GLUT_BITMAP_HELVETICA_18);
            glColor3f(0.0,1.0,1.0);
            drawstring(200.0,455.0,1.0,"P L A N E  M I R R O R");
```

```
        glFlush();
        glColor3f(0.0,1.0,1.0);
        glBegin(GL_LINES);
        glVertex2f(195.0,450.0);
        glVertex2f(300.0,450.0);
        glEnd();
        glColor3f(0.5,0.5,0.5);
        glBegin(GL_LINES);
        glVertex2f(250,400);
        glVertex2f(250,200);
        glEnd();
        GLfloat l;
        for(l=205.0;l<400.0;l+=5.0)
                shading(250.0,l);


        glColor3f(1.0,0.0,0.0);
        //object
    glBegin(GL_LINES);
        glVertex2f(200,300);
        glVertex2f(200,325);
    glVertex2f(200,325);
        glVertex2f(195,320);
        glVertex2f(200,325);
        glVertex2f(205,320);
        glEnd();
        glFlush();
        delay();

        glColor3f(1.0,1.0,0.0);
        //incident ray
```

```
glBegin(GL_LINES);
        glVertex2f(200,325);
        glVertex2f(250,300);
glVertex2f(250,300);
        glVertex2f(245,305);
        glVertex2f(250,300);
        glVertex2f(244,301);
        glVertex2f(200,325);
        glVertex2f(250,325);

        glVertex2f(250,325);
        glVertex2f(244,327);
        glVertex2f(250,325);
        glVertex2f(244,323);
        glEnd();
        glFlush();
        delay();

        glColor3f(0.0,0.0,0.0);
        //clear traces
        glBegin(GL_LINES);
glVertex2f(250,325);
        glVertex2f(244,327);
        glVertex2f(250,325);
        glVertex2f(244,323);
        glEnd();


glColor3f(0.0,1.0,0.0);
        //reflected ray
glBegin(GL_LINES);
```

```
        glVertex2f(200,325);
        glVertex2f(250,325);
        glVertex2f(225,325);
        glVertex2f(229,329);
glVertex2f(225,325);
        glVertex2f(229,321);

        glVertex2f(200,275);
        glVertex2f(250,300);

        glVertex2f(200,275);
        glVertex2f(203,280);
        glVertex2f(200,275);
        glVertex2f(205,275);

        glEnd();

        glColor3f(0.0,1.0,0.0);
         glPushAttrib(GL_ENABLE_BIT);                    //dashed line
         glLineStipple(4,0xAAAA);
         glEnable(GL_LINE_STIPPLE);
         drawOneLine(250,300,300,325);
 drawOneLine(250,325,300,325);
         glPopAttrib();

glColor3f(1.0,0.0,1.0);                                  //image
        glBegin(GL_LINES);
glVertex2f(300,325);
        glVertex2f(300,300);
        glVertex2f(295,320);
        glVertex2f(300,325);
```

```
    glVertex2f(305,320);
        glVertex2f(300,325);

        glEnd();



        glFlush();
}

void concavemirror1()
        //concave mirror concept
{
        dec();
        setFont(GLUT_BITMAP_HELVETICA_18);
        glColor3f(0.0,1.0,1.0);
        drawstring(190.0,455.0,1.0,"C O N C A V E  M I R R O R");
        glFlush();
        glColor3f(0.0,1.0,1.0);
        glBegin(GL_LINES);
        glVertex2f(185.0,450.0);
        glVertex2f(310.0,450.0);
        glEnd();
        circle(120,300,130);
        glColor3f(1.0,1.0,1.0);
        glBegin(GL_POINTS);
        glVertex2f(120.0,300.0);
        glVertex2f(185.0,300.0);
        glEnd();
        setFont(GLUT_BITMAP_HELVETICA_18);
        glColor3f(1.0,1.0,1.0);
        drawstring(120.0,290.0,1.0,"R");
```

```
          glColor3f(1.0,1.0,1.0);
          drawstring(185.0,290.0,1.0,"F");
          glFlush();

          glColor3f(1.0,1.0,0.0);
          //incident ray
     glBegin(GL_LINES);
          glVertex2f(110,350);
          glVertex2f(250,300);
     glVertex2f(250,300);
          glVertex2f(245,305);
          glVertex2f(250,300);
          glVertex2f(244,301);
          glEnd();
          glFlush();
          delay();

          glColor3f(0.0,1.0,0.0);
          //reflected ray
     glBegin(GL_LINES);
          glVertex2f(110,250);
          glVertex2f(250,300);
          glVertex2f(110,250);
          glVertex2f(113,255);
          glVertex2f(110,250);
          glVertex2f(115,250);
          glEnd();
          glFlush();
          delay();
```

```
        glColor3f(1.0,1.0,0.0);
        //incident ray
    glBegin(GL_LINES);
        glVertex2f(160,350);
        glVertex2f(240,350);
        glVertex2f(240,350);
        glVertex2f(234,353);
        glVertex2f(240,350);
        glVertex2f(234,347);

        glVertex2f(160,250);
        glVertex2f(240,250);
        glVertex2f(240,250);
        glVertex2f(234,253);
        glVertex2f(240,250);
        glVertex2f(234,247);
        glEnd();
        glFlush();
        delay();

    glColor3f(0.0,1.0,0.0);                                    //reflected ray
    glBegin(GL_LINES);
        glVertex2f(185,300);
        glVertex2f(240,350);
        glVertex2f(185,300);
        glVertex2f(195,303);
    glVertex2f(185,300);
        glVertex2f(187,305);

        glVertex2f(185,300);
        glVertex2f(240,250);
```

```
            glVertex2f(185,300);
            glVertex2f(195,297);
        glVertex2f(185,300);
            glVertex2f(187,295);
            glEnd();
            glFlush();
    }
    void concavemirror()
            //concave mirror ray diagram
    {
            dec();
            setFont(GLUT_BITMAP_HELVETICA_18);
            glColor3f(0.0,1.0,1.0);
            drawstring(190.0,455.0,1.0,"C O N C A V E  M I R R O R");
            glFlush();
            glColor3f(0.0,1.0,1.0);
            glBegin(GL_LINES);
            glVertex2f(185.0,450.0);
            glVertex2f(310.0,450.0);
            glEnd();
            circle(120,300,130);
            glColor3f(1.0,1.0,1.0);
            glBegin(GL_POINTS);
            glVertex2f(120.0,300.0);
            glVertex2f(185.0,300.0);
            glEnd();
            setFont(GLUT_BITMAP_HELVETICA_18);
            glColor3f(1.0,1.0,1.0);
            drawstring(120.0,290.0,1.0,"R");
            glColor3f(1.0,1.0,1.0);
            drawstring(185.0,290.0,1.0,"F");
            glColor3f(1.0,0.0,0.0);
            //object
        glBegin(GL_LINES);
            glVertex2f(110,300);
            glVertex2f(110,350);
        glVertex2f(110,350);
            glVertex2f(105,345);
            glVertex2f(110,350);
            glVertex2f(115,345);
            glEnd();
            glFlush();
            delay();

            glColor3f(1.0,1.0,0.0);
            //incident ray
        glBegin(GL_LINES);
            glVertex2f(110,350);
            glVertex2f(250,300);
        glVertex2f(250,300);
            glVertex2f(245,305);
            glVertex2f(250,300);
            glVertex2f(244,301);

            glVertex2f(110,350);
            glVertex2f(240,350);
            glVertex2f(240,350);
            glVertex2f(234,353);
            glVertex2f(240,350);
            glVertex2f(234,347);
            glEnd();
            glFlush();
            delay();

        glColor3f(0.0,1.0,0.0);
            //reflected ray
        glBegin(GL_LINES);
            glVertex2f(130,250);
            glVertex2f(240,350);
            glVertex2f(185,300);
            glVertex2f(195,303);
        glVertex2f(185,300);
            glVertex2f(187,305);

            glVertex2f(110,250);
            glVertex2f(250,300);

            glVertex2f(110,250);
            glVertex2f(113,255);
            glVertex2f(110,250);
            glVertex2f(115,250);
            glEnd();
            glFlush();
            delay();

            glColor3f(1.0,0.0,1.0);
            //image
        glBegin(GL_LINES);
            glVertex2f(142.941,300);
            glVertex2f(142.941,261.1765);
        glVertex2f(142.941,261.1765);
            glVertex2f(141,267);
            glVertex2f(142.941,261.1765);
            glVertex2f(144.882,267);
            glEnd();
            glFlush();
    }

    void convexmirror1()
            //convex mirror concept
    {
            dec();
            setFont(GLUT_BITMAP_HELVETICA_18);
            glColor3f(0.0,1.0,1.0);
            drawstring(190.0,455.0,1.0,"C O N V E X  M I R R O R");
            glFlush();
            glColor3f(0.0,1.0,1.0);
            glBegin(GL_LINES);
            glVertex2f(185.0,450.0);
            glVertex2f(310.0,450.0);
            glEnd();
            circle1(380,300,130);
            glColor3f(1.0,1.0,1.0);
            glBegin(GL_POINTS);
            glVertex2f(315.0,300.0);
            glVertex2f(380.0,300.0);
            glEnd();
            setFont(GLUT_BITMAP_HELVETICA_18);
            glColor3f(1.0,1.0,1.0);
            drawstring(380.0,290.0,1.0,"R");
            glColor3f(1.0,1.0,1.0);
            drawstring(315.0,290.0,1.0,"F");
            glFlush();
```

```
          glColor3f(1.0,1.0,0.0);
                              //incident ray
glBegin(GL_LINES);
          glVertex2f(110,350);
          glVertex2f(250,300);
glVertex2f(250,300);
          glVertex2f(245,305);
          glVertex2f(250,300);
          glVertex2f(244,301);
          glEnd();
          glFlush();
          delay();

          glColor3f(0.0,1.0,0.0);
                    //reflected ray
glBegin(GL_LINES);
          glVertex2f(110,250);
          glVertex2f(250,300);
          glVertex2f(110,250);
          glVertex2f(113,255);
          glVertex2f(110,250);
          glVertex2f(115,250);
          glEnd();
          glFlush();
          delay();

          glColor3f(1.0,1.0,0.0);
                    //incident ray
glBegin(GL_LINES);
          glVertex2f(160,350);
```

```
        glVertex2f(260,350);
        glVertex2f(260,350);
        glVertex2f(254,353);
        glVertex2f(260,350);
        glVertex2f(254,347);

        glVertex2f(160,250);
        glVertex2f(260,250);
        glVertex2f(260,250);
        glVertex2f(254,253);
        glVertex2f(260,250);
        glVertex2f(254,247);
        glEnd();
        glFlush();
        delay();



    glColor3f(0.0,1.0,0.0);
        //reflected ray
    glBegin(GL_LINES);
        glVertex2f(150,450);
        glVertex2f(260,350);
        glVertex2f(150,450);
        glVertex2f(156,447);
    glVertex2f(150,450);
        glVertex2f(153,445);

        glVertex2f(150,150);
        glVertex2f(260,250);
        glVertex2f(150,150);
        glVertex2f(156,153);
    glVertex2f(150,150);
        glVertex2f(153,155);
        glEnd();

        glPushAttrib(GL_ENABLE_BIT);
        //dashed line
         glLineStipple(4,0xAAAA);
         glEnable(GL_LINE_STIPPLE);
         drawOneLine(260,250,315,300);
    drawOneLine(260,350,315,300);
         glPopAttrib();
        glFlush();
        delay();
        glFlush();
}

void convexmirror()
                    //convex mirror ray diagram
{

        dec();
        setFont(GLUT_BITMAP_HELVETICA_18);
        glColor3f(0.0,1.0,1.0);
        drawstring(190.0,455.0,1.0,"C O N V E X  M I R R O R");
        glFlush();
        glColor3f(0.0,1.0,1.0);
        glBegin(GL_LINES);
        glVertex2f(185.0,450.0);
        glVertex2f(310.0,450.0);
        glEnd();
        circle1(380,300,130);
        glColor3f(1.0,1.0,1.0);
        glBegin(GL_POINTS);
        glVertex2f(315.0,300.0);
        glVertex2f(380.0,300.0);
        glEnd();
        setFont(GLUT_BITMAP_HELVETICA_18);
        glColor3f(1.0,1.0,1.0);
        drawstring(380.0,290.0,1.0,"R");
        glColor3f(1.0,1.0,1.0);
        drawstring(315.0,290.0,1.0,"F");

        glColor3f(1.0,0.0,0.0);
                        //object
    glBegin(GL_LINES);
        glVertex2f(110,300);
        glVertex2f(110,350);
    glVertex2f(110,350);
        glVertex2f(105,345);
        glVertex2f(110,350);
        glVertex2f(115,345);
        glEnd();
        glFlush();
        delay();

        glColor3f(1.0,1.0,0.0);
                            //incident ray
    glBegin(GL_LINES);
        glVertex2f(110,350);
        glVertex2f(250,300);
    glVertex2f(250,300);
        glVertex2f(245,305);
        glVertex2f(250,300);
        glVertex2f(244,301);

        glVertex2f(110,350);
        glVertex2f(260,350);
        glVertex2f(260,350);
        glVertex2f(254,353);
        glVertex2f(260,350);
        glVertex2f(254,347);
        glEnd();
        glFlush();
        delay();

    glColor3f(0.0,1.0,0.0);
                //reflected ray
    glBegin(GL_LINES);
        glVertex2f(150,450);
        glVertex2f(260,350);
        glVertex2f(150,450);
        glVertex2f(156,447);
    glVertex2f(150,450);
        glVertex2f(153,445);

        glVertex2f(110,250);
        glVertex2f(250,300);
        glVertex2f(110,250);
        glVertex2f(113,255);
        glVertex2f(110,250);
        glVertex2f(115,250);
        glEnd();
```

```
            glPushAttrib(GL_ENABLE_BIT);
                          //dashed line
             glLineStipple(4,0xAAAA);
             glEnable(GL_LINE_STIPPLE);
             drawOneLine(250,300,390,350);
        drawOneLine(260,350,315,300);
             glPopAttrib();
            glFlush();
            delay();

            glColor3f(1.0,0.0,1.0);
                                       //image
        glBegin(GL_LINES);
            glVertex2f(296.667,300);
            glVertex2f(296.667,316.667);
            glVertex2f(294,313);
            glVertex2f(296.667,316.667);
            glVertex2f(300.33,313);
            glVertex2f(295.667,316.667);
            glEnd();

            glFlush();
    }

    void mainmenu()
                    //initial menu
    {
            glClear(GL_COLOR_BUFFER_BIT);
            glBegin(GL_POLYGON);
            glColor3f(0.7f,0.2f,0.2f);
        glVertex2i(190,315);
            glVertex2i(65,315);
            glVertex2i(65,285);
        glVertex2i(190,285);
            glEnd();

            glBegin(GL_POLYGON);
            glColor3f(0.7f,0.2f,0.2f);
        glVertex2i(420,315);
            glVertex2i(295,315);
            glVertex2i(295,285);
        glVertex2i(420,285);
            glEnd();

            setFont(GLUT_BITMAP_HELVETICA_18);
            glColor3f(1.0,1.0,1.0);
            drawstring(120.0,455.0,1.0,"* * * * * * * * * R E F L E C T I O N * * * *
    * * * * *");

            glColor3f(1.0,1.0,1.0);
            drawstring(95.0,295.0,1.0,"  CONCEPT");

            glColor3f(1.0,1.0,1.0);
            drawstring(95.0,265.0,1.0,"  PRESS C/c");


            glColor3f(1.0,1.0,1.0);
            drawstring(325.0,295.0,1.0,"  RAY DIAGRAM");

            glColor3f(1.0f,1.0f,1.0f);
            drawstring(325.0,265.0,1.0,"PRESS R/r");
            glColor3f(1.0f,1.0f,1.0f);
```

```
            drawstring(200.0,110.0,1.0," PRESS N TO EXIT");
            glFlush();
}

void key(char *s)                                    // text for keyboard interaction

{

            glColor3f(0.0,0.0,0.0);
            glBegin(GL_POLYGON);
            glVertex2f(0,0);
            glVertex2f(0,135);
            glVertex2f(500,135);
            glVertex2f(500,0);
            glEnd();
            glFlush();



            glColor3f(1.0f,1.0f,1.0f);
            setFont(GLUT_BITMAP_HELVETICA_18);
            drawstring(360.0,130.0,1.0,"DO U WISH TO CONTINUE?");
            glColor3f(1.0f,1.0f,1.0f);
            drawstring(360.0,110.0,1.0,s);
            glFlush();
}
void concept()
            //menu of concept
{
            setFont(GLUT_BITMAP_HELVETICA_18);
            glColor3f(1.0,0.0,1.0);
            drawstring(90.0,400.0,1.0,"P A R A L L E L  M I R R O R");
```

```
        glColor3f(1.0,0.0,1.0);
        glBegin(GL_LINES);
        glVertex2f(85.0,395.0);
        glVertex2f(210.0,395.0);
        glEnd();

        glColor3f(1.0,1.0,1.0);
        drawstring(50.0,375.0,1.0,"1)      Rays  normal  to  the mirror  gets");

        glColor3f(1.0,1.0,1.0);
        drawstring(100.0,365.0,1.0,"  reflected  normally");

        glColor3f(1.0,1.0,1.0);
        drawstring(50.0,345.0,1.0,"2)      Rays  incident  at  an  angle follows");

        glColor3f(1.0,1.0,1.0);
        drawstring(100.0,335.0,1.0,"LAW  OF  REFLECTION , i.e.,");
        glColor3f(1.0,1.0,1.0);
        drawstring(65.0,320.0,1.0,"angle  of  INCIDENCE  =  angle  of
REFLECTION");

        glColor3f(1.0,0.0,1.0);
        drawstring(90.0,250.0,1.0,"S P H E R I C A L  M I R R O R");

        glColor3f(1.0,0.0,1.0);
        glBegin(GL_LINES);
        glVertex2f(85.0,245.0);
        glVertex2f(210.0,245.0);
        glEnd();
        glColor3f(1.0,1.0,1.0);
        drawstring(50.0,225.0,1.0,"1)      Rays  coming from infinity , parallel");

        glColor3f(1.0,1.0,1.0);
        drawstring(70.0,215.0,1.0,"to  the  principal  axis  after  reflection");

        glColor3f(1.0,1.0,1.0);
        drawstring(70.0,205.0,1.0,"passes  or  appears  to  pass  through  FOCUS");

        glColor3f(1.0,1.0,1.0);
        drawstring(50.0,185.0,1.0,"2)      Rays  incident  at  P(pole)  follows");

        glColor3f(1.0,1.0,1.0);
        drawstring(100.0,175.0,1.0,"LAW  OF  REFLECTION , i.e.,");
        glColor3f(1.0,1.0,1.0);
        drawstring(65.0,160.0,1.0,"angle  of  INCIDENCE  =  angle  of
REFLECTION");
        glFlush();
}

void mykeyboard(unsigned char key,int x,int y)
        //for keyboard interaction
{
        if(key=='1' || key=='r' || key=='R')
        {
                draw(180.0,305.0);
                text(200.0);
                glColor3f(1.0,1.0,1.0);
                drawstring(120.0,455.0,1.0,"*  *  *  *  *  *  *  *  *  R A Y   D I A G R A
M  *  *  *  *  *  *  *  *  *");
                glFlush();
        }
        else if(key=='y' || key=='Y' || key=='c' || key=='C')
        {
                draw(295.0,420.0);
                text(315.0);
                glColor3f(1.0,1.0,0.0);
                drawstring(135.0,455.0,1.0,"*  *  *  *  *  *  *  *  *  C O N C E P T *  *  *
*  *  *  *  *  *");
                concept();
                glFlush();
        }
        else if(key=='N' || key=='n' || key=='0')
        {
                exit(0);
        }


}

void myMouse(int btn, int state, int x, int y)
        //for mouse interaction
{

        if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN )
        {
                x=x/2;
                y=(1000-y)/2;
                        if ((x>=180 && x<=305) && (y>=420 && y<=445 ))

                        {
                                planemirror();
        key("PRESS 1/0");

    }

    if((x>=180 && x<=305) && (y>=380 && y<=410))
    {
        concavemirror();
        key("PRESS 1/0");
    }
    if((x>=180 && x<=305) && (y>=340 && y<=370 ))
    {

        convexmirror();
        key("PRESS 1/0");
    }

    if(((x>=180 && x<=305) || (x>=295 && x<=420)) && (y>=220
&& y<=250))
    {
            mainmenu();
    }

    if ((x>=295 && x<=420) && (y>=420 && y<=445 ))

    {
        planemirror1();
        key("PRESS Y/N");

    }

    if((x>=295 && x<=420)&& (y>=380 && y<=410))
```

```
                        {
                                concavemirror1();
                                key("PRESS Y/N");
                        }
                        if((x>=295 && x<=420) && (y>=340 && y<=370 ))
                        {

                                convexmirror1();
                                key("PRESS Y/N");
                        }
        }

}


void display(void)
        //display function
{

        glClear(GL_COLOR_BUFFER_BIT);
        mainmenu();
}


void main(int argc,char **argv)
        //main function
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowPosition(0,0);
        glutInitWindowSize(1000,1000);
        glutCreateWindow("reflection");
    glutDisplayFunc(display);
        glutMouseFunc(myMouse);
        glutKeyboardFunc(mykeyboard);
        myInit();
        glutMainLoop();
}
```

Snapshots

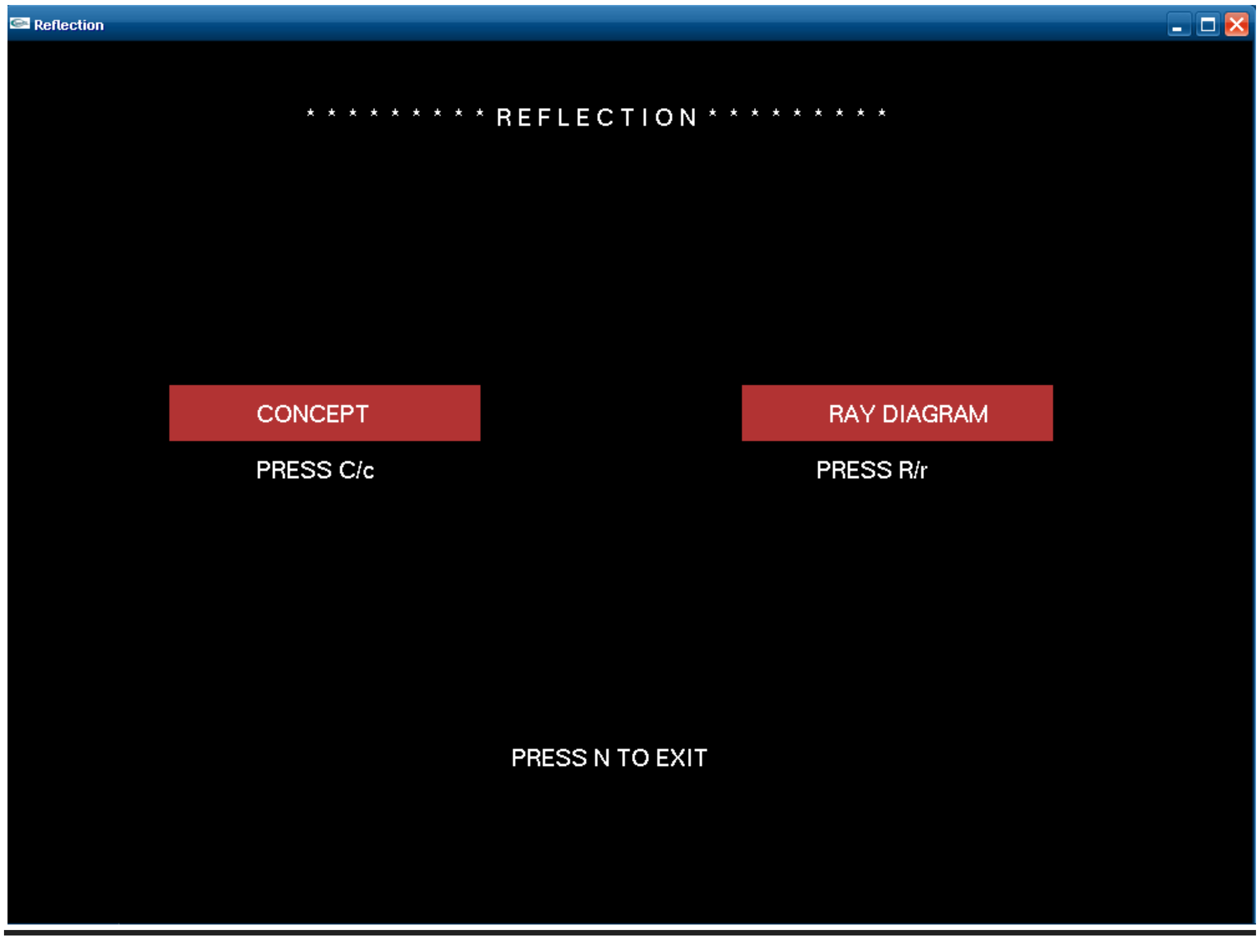
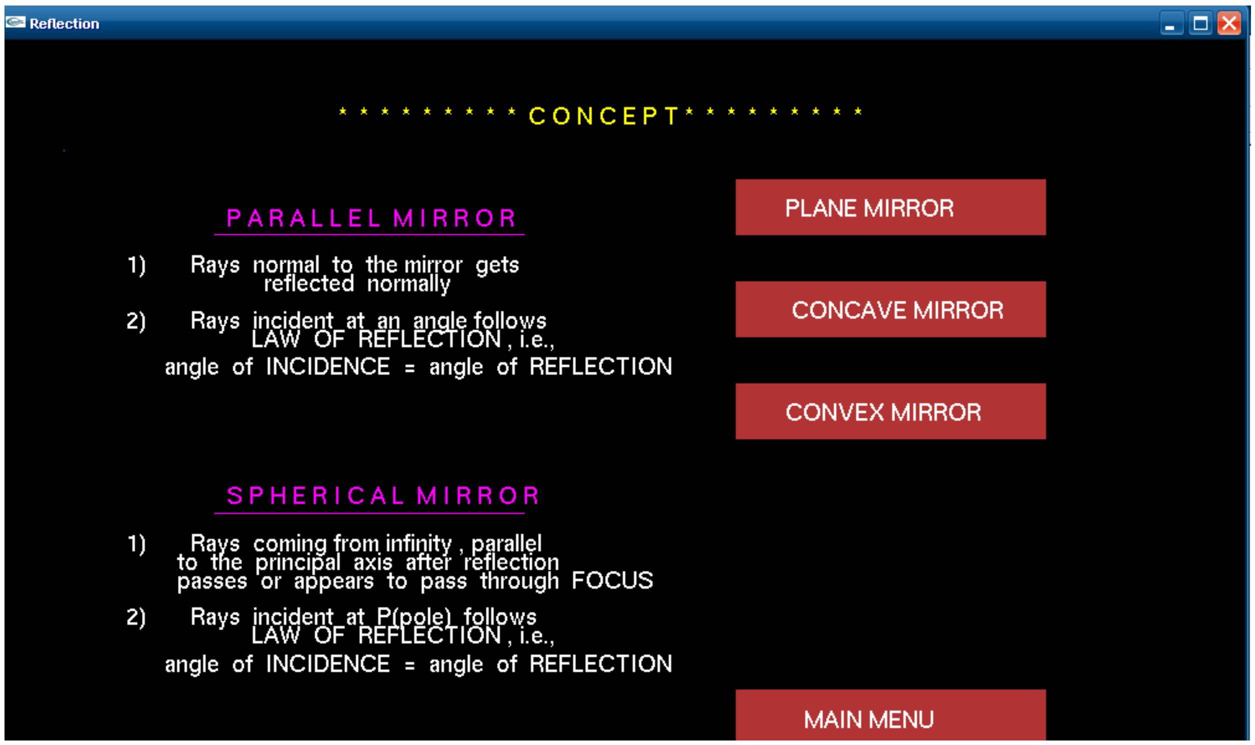


Fig 1: The above figure shows the main menu



Reflection

* * * * * * * * * CONCEPT * * * * * * * * *

PARALLEL MIRROR

1) Rays normal to the mirror gets reflected normally

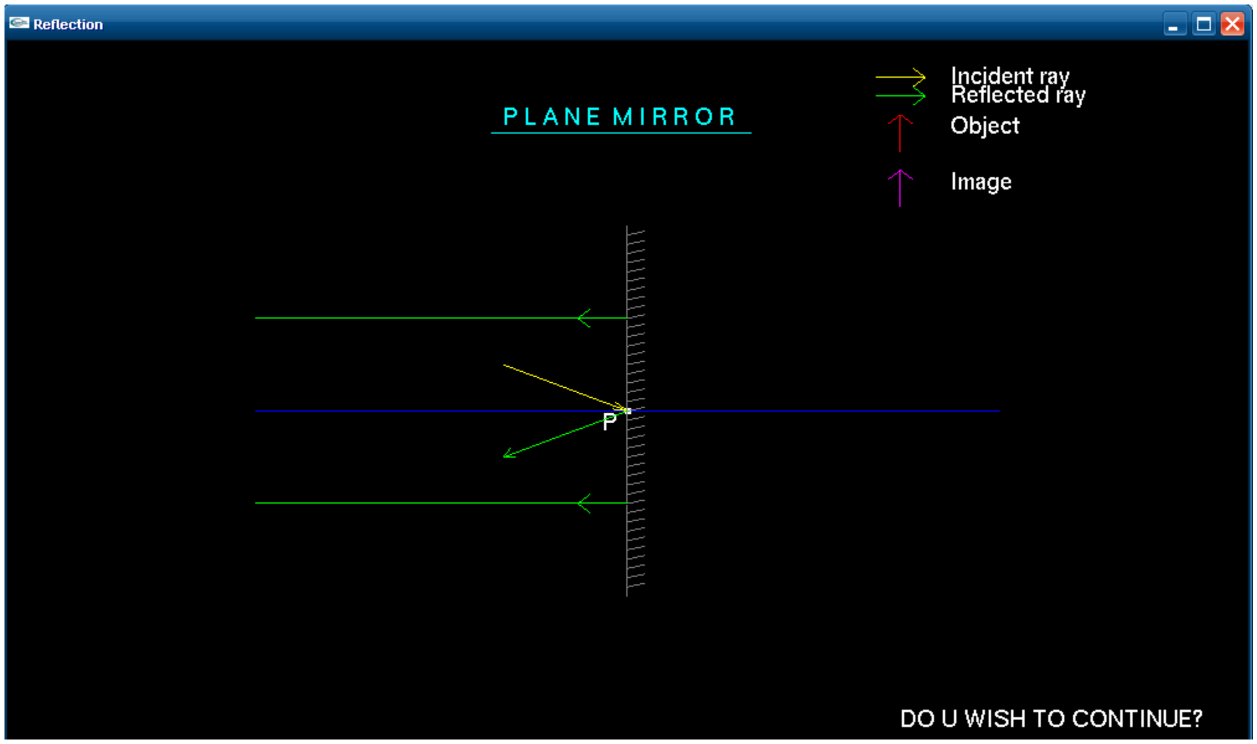
2) Rays incident at an angle follows LAW OF REFLECTION, i.e., angle of INCIDENCE = angle of REFLECTION

SPHERICAL MIRROR

1) Rays coming from infinity, parallel to the principal axis after reflection passes or appears to pass through FOCUS

2) Rays incident at P(pole) follows LAW OF REFLECTION, i.e., angle of INCIDENCE = angle of REFLECTION

Fig 2: The above figure shows the concept and its menu



Reflection

PLANE MIRROR

DO U WISH TO CONTINUE?

Fig 3: the concept of plane mirror

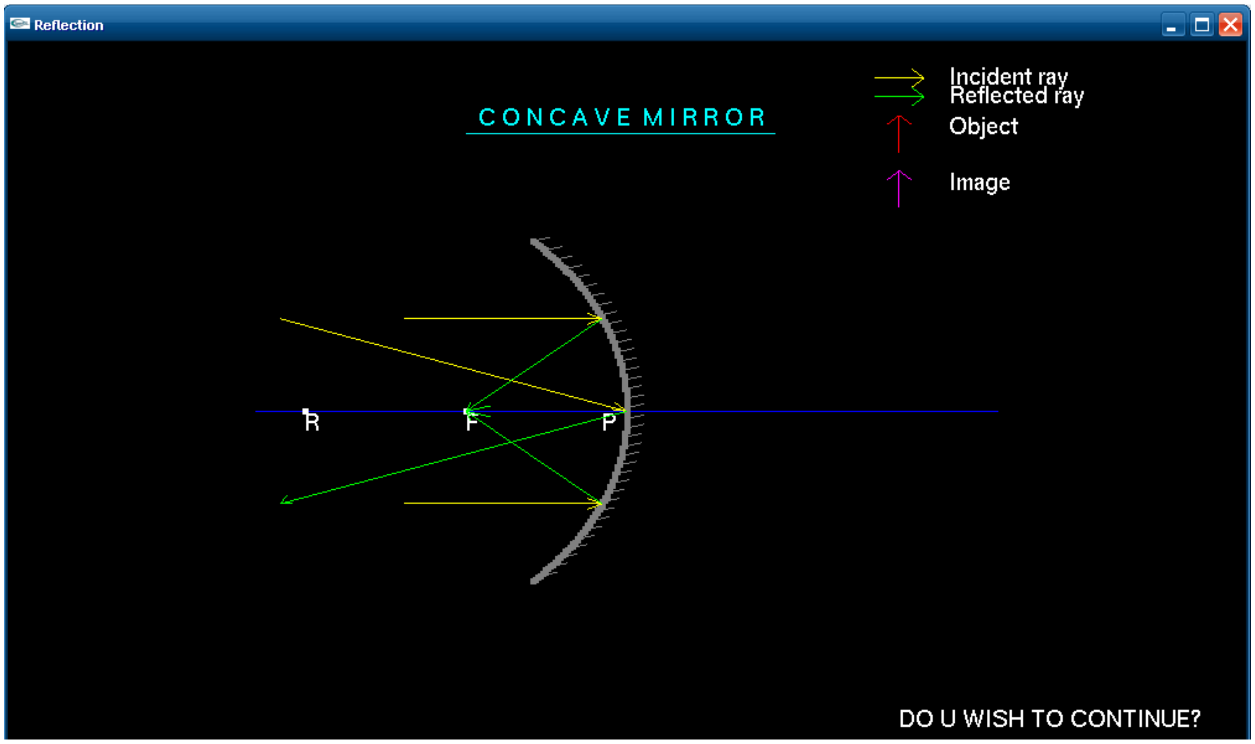


Fig 4: concept of concave mirror.

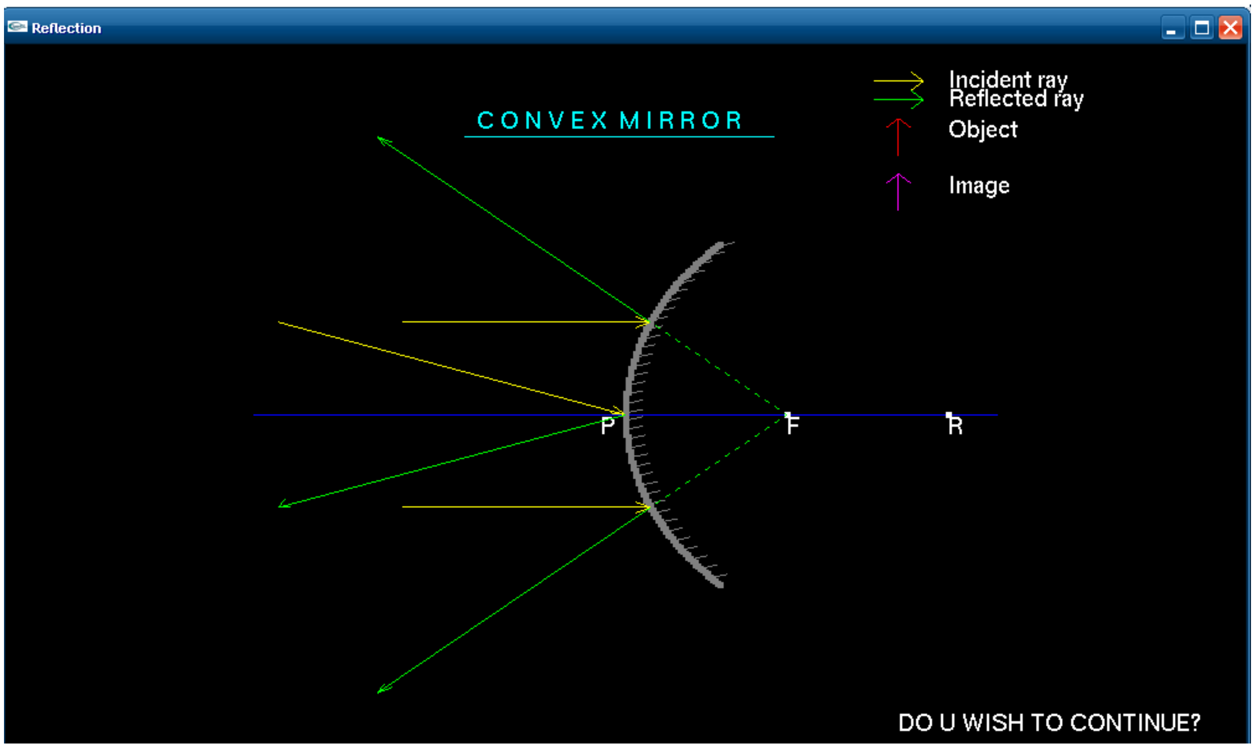


Fig 5: concept of convex mirror.

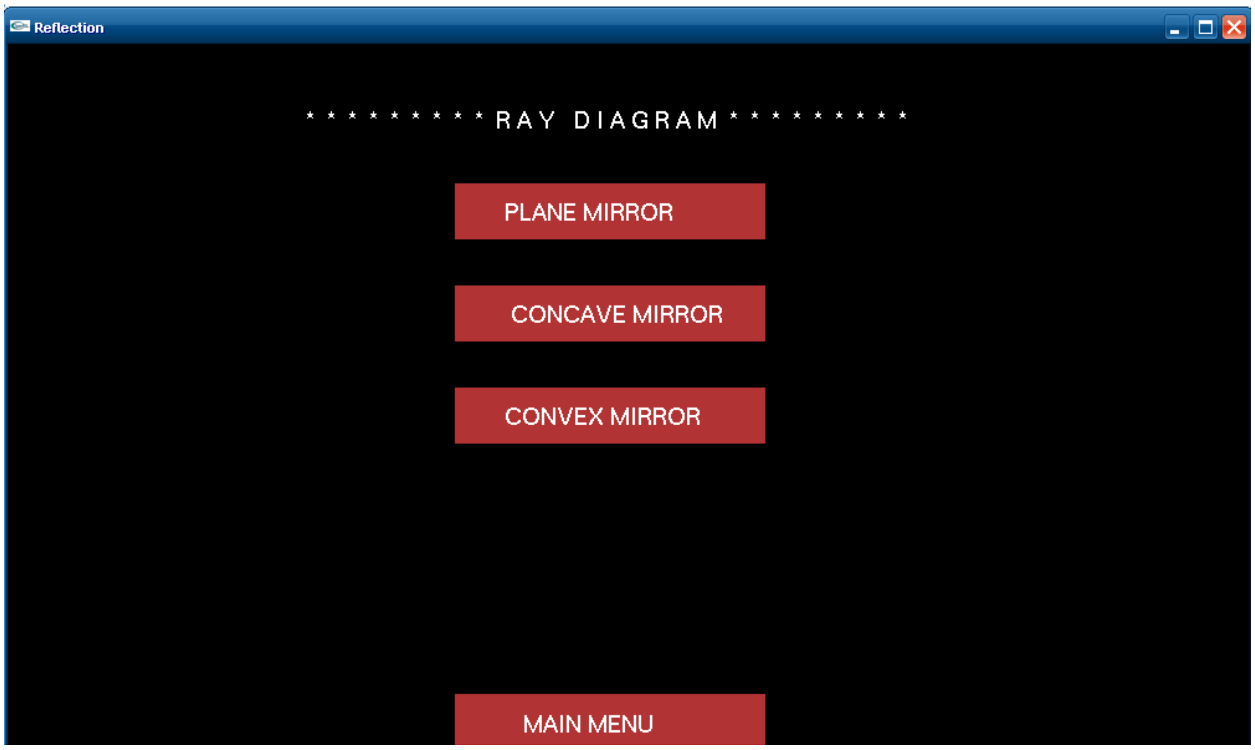


Fig 6: Menu for ray diagram.

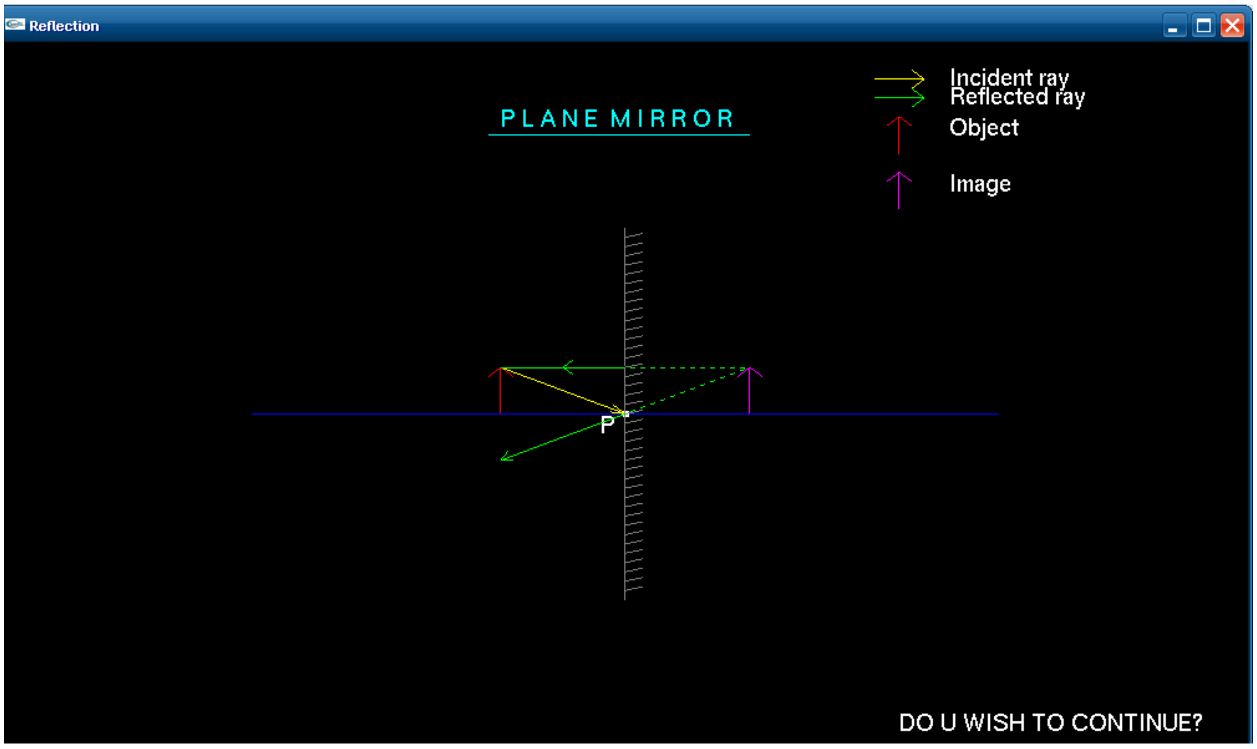


Fig 7: Ray Diagram for plane mirror

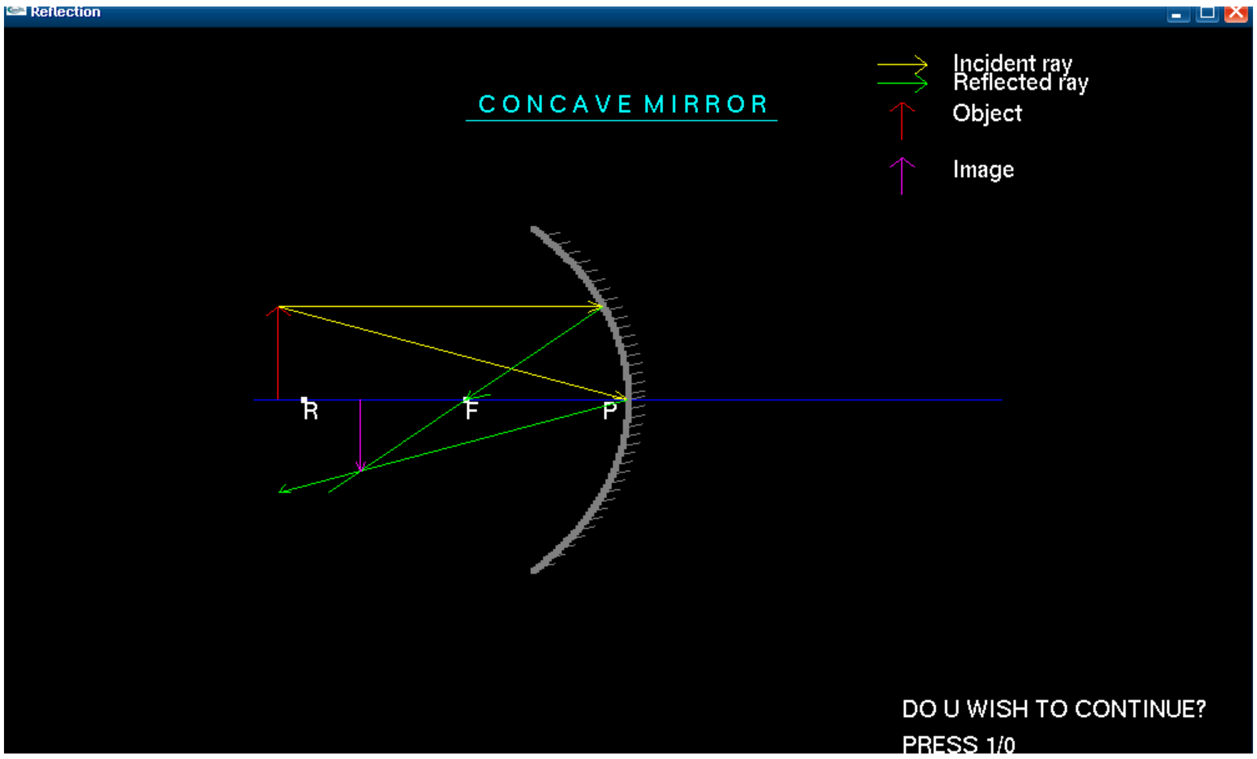


Fig 8: Ray Diagram for concave mirror.

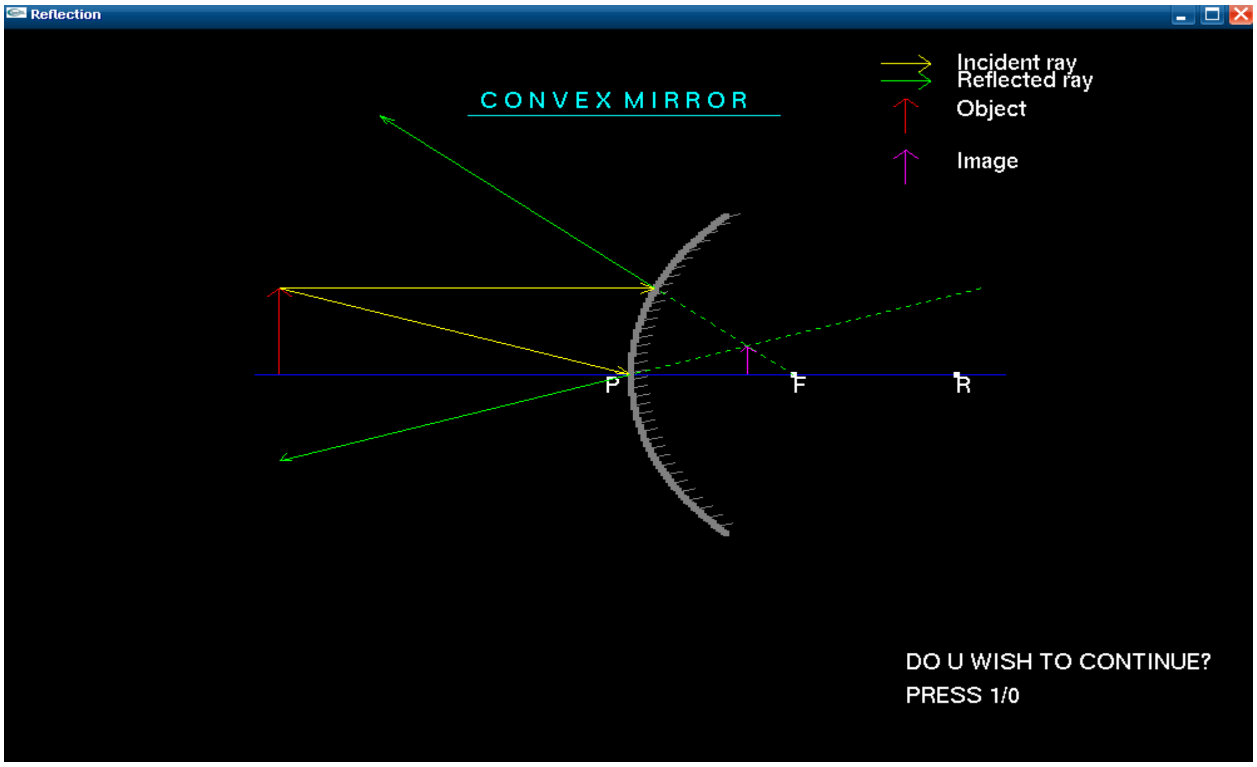


Fig 9: Ray Diagram for convex mirror

Testing

- **TEST CASES**

  Software testing is a critical element of the ultimate review of specification design and coding. Testing of software leads to the uncovering of errors in the software functional and performance requirements are met. Testing also provides a good indication of software reliability and software quality as a whole. The result of different phases of testing are evaluated and then compared with the expected results. If the errors are uncovered they are debugged and corrected.

- **TEST PLANS:**

  In this test plan all major activities are described below:

  - Unit Testing:Unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming a unit could be an entire module but is more commonly an individual function or procedure. In object-oriented programming a unit is often an entire interface, such as a class, but could be an individual method.Unit tests are created by programmers or occasionally by white box testers during the development process.

  - System Testing: System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

  - Integration Testing. Integration testing (sometimes called Integration and Testing, abbreviated "I&T") is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing.

CONCLUSION

We have shown computer graphics is a method of image formation that should be related to classical method of image formation. We have also introduced OpenGL API and the applied basic concepts. The historical development of graphics APIs and graphical method illustrates the starting in 3 dimensions.

The interactive aspects make the field of computer graphics exciting and fun. Although our API, OpenGL, is independent of any operating or window system, we recognize that any program must have at least minimal interaction with the rest of the computer system. We handled simple interactions by using a simple tool kit, GLUT, whose API provides the necessary additional functionality, without being dependent on a particular operating or window system. From the application programmer's perspective, various characteristics of interactive graphics are shared by most systems. We see the graphic part of the system as a server, consisting of a raster display, a keyboard, and a pointing device. Thus interactive computer graphics is a powerful tool with unlimited applications.

Hence, we conclude that OpenGL remains the standard programmer's interface both for writing application programs and developing high-level products of multiplatform applications. OpenGL supports application ranging from large scientific visualizations to cell phone games.

BIBLIOGRAPHY

Books referred

1. Edward Angel - Interactive Computer Graphics A Top-Down Approach with OpenGL, 5th Edition, Addison-Wesley, 2008.
2. F. S. Hill - Computer Graphics Using OpenGL – 2nd Edition, Pearson Education, 2001,
3. Physics Light Reflection and Refraction Grade-11

Running the Code

Compiling on Mac OS :

```
$ gcc -o Reflection Reflection.cc -framework GLUT -framework OpenGL
```

Running on Mac OS:

```
$ ./Reflection
```