**CHAPTER 1**

**1.Introduction**

***Computer graphics*** are graphics created using computers and, more generally, the representation and manipulation of pictorial data by a computer.

The development of computer graphics has made computers easier to interact with and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry.

The term computer graphics includes almost everything on computers that is not text or sound. Today nearly all computers use some graphics and users expect to control their computer through icons and pictures rather than just by typing. The term Computer Graphics has several meanings:

•The representation and manipulation of pictorial data by a compute.

•The various technologies used to create and manipulate such pictorial data.

•The images so produced, and

•The sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content see study of computer graphics.

There are two types of computer graphics – raster and vector. The raster graphic is composed of pixels and vector graphic is composed of drawing paths. It is important to understand the difference between these two types before you choose the graphics format to save the barcode image.

Today computers and computer-generated images touch many aspects of our daily life. 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.

***1.1 Computer Graphics***

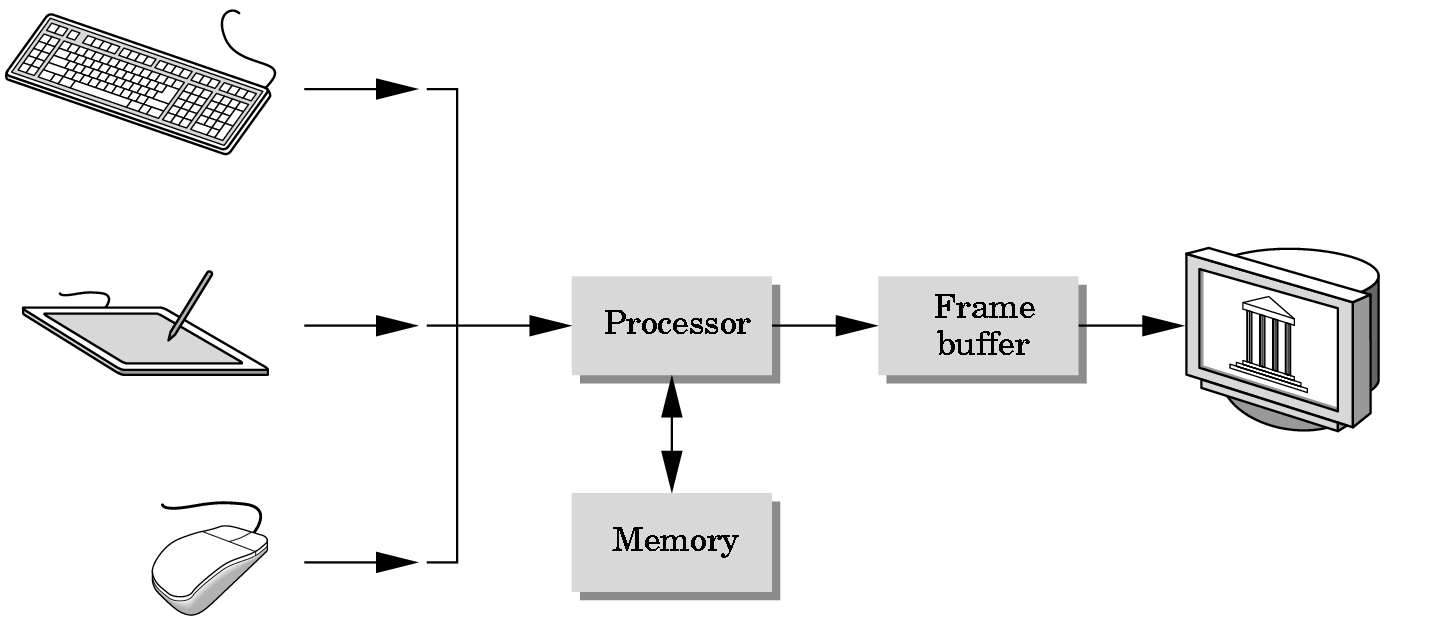
The term computer graphics includes almost everything on computers that is not text or sound. Today nearly all computers use some graphics and users expect to control their computer through icons and pictures rather than just by typing. The term Computer Graphics has several meanings:

* The representation and manipulation of pictorial data by a computer
* The various technologies used to create and manipulate such pictorial data
* The images so produced, and
* The sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content

Today computers and computer-generated images touch many aspects of our daily life. 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.

A computer graphics system is a computer system, as such; it must have all the components of a general-purpose computer system. The high level view of a graphics system as shown in the block diagram in Fig 1.1.1 contains five major elements

Fig 1.1.1 A Graphics System.



Input devices

Output Device

***1.2 OpenGL Technology***

OpenGL is strictly defined as “a software interface to graphics hardware.” In essence, it is a 3D graphics and modeling library that is highly portable and very fast.

Using OpenGL, you can create elegant and beautiful 3D graphics with exceptional visual quality. The greatest advantage to using OpenGL is that it is orders of magnitude faster than a ray tracer or software-rendering engine. Initially, it used algorithms carefully developed and optimized by Silicon Graphics, Inc. (SGI), an acknowledged world leader in computer graphics and animation. Over time, OpenGL has evolved as other vendors have contributed their expertise and intellectual property to develop high-performance implementations of their own.

OpenGL is a software interface to graphics hardware. This interface consists of about 120 distinct commands, which you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed to work efficiently even if the computer that displays the graphics you create isn't the computer that runs your graphics program. This might be the case if you work in a networked computer environment where wires capable of carrying digital data connect many computers to one another. In this situation, the computer on which your program runs and issues OpenGL drawing commands is called the client, and the computer that receives those commands and performs the drawing

is called the server. The format for transmitting OpenGL commands (called the protocol) from the client to the server is always the same, so OpenGL programs can work across a network even if the client and server are different kinds of computers. If an OpenGL program isn't running across a network, then there's only one computer, and it is both the client and the server. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. Computer graphics deals with all aspects of creating images with a computer.The processing of geometry of the objects to obtain an image can be employed using the four major steps as shown in Fig 1.2.1

Application: The object is an artist’s rendition of the sun for an animation to be shown in a domed environment (planetarium)

Software: Maya for modeling and rendering but Maya is built on top of OpenGL

Hardware: PC with graphics card for modeling and rendering.

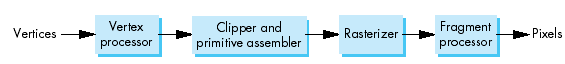


Fig 1.2.1 Graphics pipeline.

Fig 1.2.2 shows the organization of the libraries for an X Window System environment. For this Window System, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems

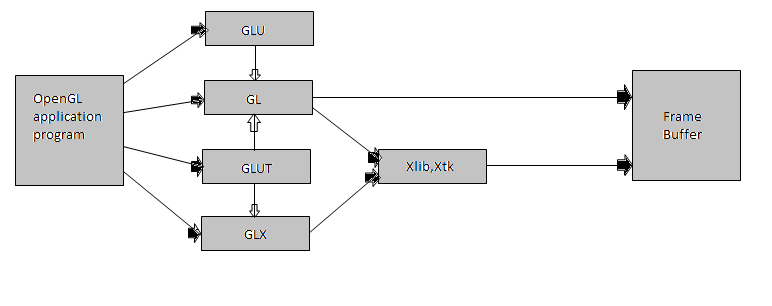


Fig 1.2.2 Library Organization

***1.3* Microsoft Visual Studio**

Microsoft Visual Studio is an Integrated Development Environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer, class designer, and database schema designer. It allows plug-ins to be added that enhance the functionality at almost every level - including adding support for source control systems (like Subversion and Visual SourceSafe) to adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).

Visual Studio supports languages by means of language services, which allow any programming language to be supported (to varying degrees) by the code editor and debugger, provided a language-specific service has been authored. Built-in languages include C/C++ (via Visual C++), VB.NET (via Visual Basic .NET), and C# (via Visual C#). Support for other languages such as Chrome, F#, Python, and Ruby among others has been made available via language services which are to be installed separately. It also supports XML/XSLT, HTML/XHTML, JavaScript and CSS. Language-specific versions of Visual Studio also exist which provide more limited language services to the user. These individual packages are called Microsoft Visual Basic, Visual J#, Visual C#, and Visual C++.

**CHAPTER 2**

**Requirements**

**2.1:Software Requirements:**

* Operating System : window 98/higher, window XP with dos environment/windows vista.
* Language Tool : OpenGL
* Compiler : GNU GCC Compiler /C++ complier.
* Libraries : Supporting glut32.h, opengl32.h & glu32.h .
* Documentation Tool : Visual C++ 6 or higer versions like 2008.

**2.2:Hardware Requirements:**

* Processor : Intel 386 onwards Compatible Hardware.
* RAM : 16Mb RAM
* Hard Disk : use in KB
* Monitor : EGVGA Compatible
* Keyboard : Standard 101 key Keyboard
* Mouse(ps/2)

**CHAPTER 3**

**Over view**

The Computer Graphics is one of the most effective and commonly used methods to communicate the processed information to the user. It displays the information in the form of graphics objects such as pictures, charts, graphs and diagram instead of simple text.

In computer graphics, pictures or graphics objects are presented as a collection of discrete picture elements called **pixels**. The pixel is the smallest addressable screen element

Computer graphics today is largely interactive: The user controls the contents structure, and appearance of objects and their displayed images by using input devices, such as a keyboard, mouse, or touch-sensitive panel on the screen

Computer graphics concerns with the pictorial synthesis of real or imaginary objects from their computer based models, where as the related field of image processing treats the converse process ,the analysis of scenes ,or the reconstruction of models of 2D or 3D objects from their pictures.

The image processing can be classified as

* Image enhancement.
* Pattern detection and recognition
* Scene analysis and computer vision.

The image enhancement deals with the improvement in the image quality by eliminating noise or by increasing image contrast. Pattern detection and recognition deals with the detection and clarification of standard patterns

and finding deviations from these patterns .The optical character recognition (OCR) technology is an practical example for pattern detection & recognition. Scene analysis deals with the recognition and reconstruction of 3D model of scene from several 2D images.

**3.1: Advantages of Interactive Graphics**

It provide a tool for producing pictures not only of concrete, ”real-world” objects but also of abstract ,synthesis of objects.

* It as an ability to show moving pictures, and thus it is possible to produce animations.
* With the use of interactive graphics we can control the movement of an object. The interactive graphics provides tool called motion dynamics. With this tool user can move and tumble objects with respect to stationary observer, or he can make object stationary and the viewer moving around them.
* Interactive graphics provides facility called update dynamics.
* With the recent development of digital signal processing (DSP) and audio synthesis chip the interactive graphics can now provide audio feed back along with the graphical feed backs to make the simulated environment even more realistic

**3.2: Areas of Application Computer Graphics**

* User interfaces
* Plotting of graphs and charts
* Office automation and Desktop publishing
* Computer aided Drafting and designs
* Process control
* Cartography
* Gamming

**CHAPTER 3**

**3. Interface**

Interactive computer graphics opens up a myriad of applications, ranging from interactive design of buildings, to control of large systems through graphical interfaces, to virtually reality systems, to computer games. There are varieties of devices available for interaction which can be considered in two different perspectives: the way that the physical devices can be described by their real-world properties, and the way that these devices appear to the application program.

***3.1 Interface***

Input and Interaction form the most important part of computer graphics. The mouse and the keyboard are used to interact with the application. A Mouse click or a key press acts as a trigger. The mouse function used is…

void mouse(int btn, int state, int x, int y)

The mouse function has actions defined only for particular regions on the screen. The co- ordinates of the click are checked and if they match, the corresponding actions are executed. Left clicking on the item inserts the item to the stack and right click displays the menu on the screen. Two flags are used to differentiate the startup screen with the queue display screen.

void myKey(unsignedchar key,int x,int y)

The press of ‘Enter’ key takes us to the next screen & ‘ESC’ exits the program.

**Source code**

#include<stdio.h>

//#include<GL.h>

#include<stdlib.h>

#include<GL/glut.h>

#include<math.h>

#include<windows.h>

void \*font;

void \*currentfont;

int fontType,flag=0;

int in\_about;

int enter;

float i,j;

float x2,y2,r,r1=80,r2=5.0,r3=27.5,r4=6,r5=6,x,y,angle,angle\_radians,r6=23,r7=2,r8=3,r9=6,r10=18,r11=10,r12=26,r13=14;

int k=0;

int p=0;

float toRAD = 57.2957795;

void conecting\_pipe();

/\* This function introduces delay in action by doing nothing in a loop \*/

void delay(int x)

{

int i,j;

for(i=0;i<x;i++)

{

for(j=0;j<i\*1000;j++)

;

}

}

void fig1()

{

glBegin(GL\_POINTS);

glVertex2f(255,138);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(265,158);

glVertex2f(245,158);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(277,135);

glVertex2f(263,118);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(231,133);

glVertex2f(247,118);

glEnd();

}

//glClearColor(0.0,0.0,0.0,0.0);

void fig2()

{

//glColor3f(0.7,0.4,0.3);

//glDisable(GL\_DEPTH\_TEST);

glBegin(GL\_POINTS);

glVertex2f(255,138);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(275,150);

glVertex2f(275,126);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(250,115);

glVertex2f(234,130);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(253,160);

glVertex2f(236,149);

glEnd();

}

void fig3()

{

glBegin(GL\_POINTS);

glVertex2f(255,138);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(266,117);

glVertex2f(244,117);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(277,144);

glVertex2f(261,160);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(249,159);

glVertex2f(233,144);

glEnd();

}

void fig4()

{

glBegin(GL\_POINTS);

glVertex2f(255,138);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(235,128);

glVertex2f(235,148);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(276,144);

glVertex2f(257,161);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(255,138);

glVertex2f(257,115);

glVertex2f(275,127);

glEnd();

}

/\*

void rot4()

{

glDisable(GL\_DEPTH\_TEST);

glColor3f(0.7,0.4,0.3);

fig4();

glColor3f(0.0,0.0,0.0);

fig4();

glColor3f(0.3,0.1,0.1);

fig1();

glEnable(GL\_DEPTH\_TEST);

}

void rot3()

{

glDisable(GL\_DEPTH\_TEST);

glColor3f(0.7,0.4,0.3);

fig3();

glColor3f(0.0,0.0,0.0);

fig3();

glColor3f(0.3,0.1,0.1);

fig4();

glEnable(GL\_DEPTH\_TEST);

flag=4;

}

void rot2()

{

glDisable(GL\_DEPTH\_TEST);

glColor3f(0.7,0.4,0.3);

fig2();

glColor3f(0.0,0.0,0.0);

fig2();

glColor3f(0.3,0.1,0.1);

fig3();

glEnable(GL\_DEPTH\_TEST);

flag=3;

}\*/

void rot1()

{

glDisable(GL\_DEPTH\_TEST);

glColor3f(0.0,0.0,0.0);

fig1();

glFlush();

glColor3f(0.3,0.1,0.1);

fig1();

glFlush();

//delay(1000);

glColor3f(0.0,0.0,0.0);

fig1();

//glFlush();

//delay(1000);

glColor3f(0.3,0.1,0.1);

fig2();

glFlush();

//delay(1000);

glColor3f(0.0,0.0,0.0);

fig2();

//glFlush();

glColor3f(0.3,0.1,0.1);

fig3();

glFlush();

glColor3f(0.0,0.0,0.0);

fig3();

glFlush();

glColor3f(0.3,0.1,0.1);

fig4();

glFlush();

/\*glColor3f(0.0,0.0,0.0);

fig4();

glFlush();

\*/

//delay(1000);

//glEnable(GL\_DEPTH\_TEST);

//flag=2;

}

void turbine()

{

glLineWidth(6.0);

glBegin(GL\_LINE\_LOOP);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 255 + r6\* (float)cos(angle\_radians);

y = 138 + r6\* (float)sin(angle\_radians);

glColor3f(0.3,0.1,0.1);

glVertex2f(x,y);

}

glEnd();

glLineWidth(6.0);

glBegin(GL\_LINE\_STRIP);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 255 + r12\* (float)cos(angle\_radians);

y = 138 + r12\* (float)sin(angle\_radians);

glColor3f(0.3,0.1,0.1);

glVertex2f(x,y);

}

glEnd();

glPointSize(6);

glBegin(GL\_POINTS);

glVertex2f(255,138);

glEnd();

fig4();

}

void generator()

{

glColor3f(0.5,0.5,0.9);

glBegin(GL\_POLYGON);

glVertex3i(338,120,0);

glColor3f(0.2,0.2,0.6);

glVertex3i(338,150,0);

glVertex3i(343,150,0);

glVertex3i(343,120,0);

glEnd();

glColor3f(0.5,0.5,0.9);

glBegin(GL\_POLYGON);

glVertex3i(351,120,0);

glColor3f(0.2,0.2,0.6);

glVertex3i(351,150,0);

glVertex3i(356,150,0);

glVertex3i(356,120,0);

glEnd();

glColor3f(0.5,0.5,0.9);

glBegin(GL\_POLYGON);

glVertex3i(364,120,0);

glColor3f(0.2,0.2,0.6);

glVertex3i(364,150,0);

glVertex3i(369,150,0);

glVertex3i(369,120,0);

glEnd();

glColor3f(0.3,0.1,0.1);

glBegin(GL\_POLYGON);

glVertex3i(375,144,0);

glVertex3i(277,144,0);

glVertex3i(277,137,0);

glVertex3i(375,137,0);

glEnd();

glLineWidth(6.0);

glBegin(GL\_LINE\_STRIP);

glColor3f(0.6,0.6,0.6);

glVertex2f(330,170);

glVertex2f(385,170);

glVertex2f(385,111);

glVertex2f(330,111);

glEnd();

glBegin(GL\_POLYGON);

glColor3f(0.2,0.2,0.6);

glVertex3i(330,111,0);

glVertex3i(330,120,0);

glColor3f(0.5,0.5,0.9);

glVertex3i(375,120,0);

glVertex3i(375,111,0);

glEnd();

}

void connecting\_pipe()

{

glLineWidth(3.0);

glColor3f(0.6,0.3,0.2);

glBegin(GL\_LINES);

glVertex2i(150,104);

glVertex2i(180,104);

glVertex2i(150,108);

glVertex2i(180,108);

glEnd();

glBegin(GL\_LINES);

glVertex2i(185,100);

glVertex2i(185,50);

glVertex2i(185,50);

glVertex2i(224,50);

glVertex2i(189,100);

glVertex2i(189,54);

glVertex2i(189,54);

glVertex2i(224,54);

glEnd();

}

void points()

{

glPointSize(4);

glColor3f(0.4,0.4,0.4);

glBegin(GL\_POINTS);

glVertex2f(243,134);

glVertex2f(249,128);

glVertex2f(246,122);

glVertex2f(256,115);

glVertex2f(251,110);

glVertex2f(264,108);

glVertex2f(272,104);

glVertex2f(242,110);

glVertex2f(247,100);

glVertex2f(256,104);

glVertex2f(230,100);

glVertex2f(235,105);

glVertex2f(239,110);

glVertex2f(264,104);

glVertex2f(249,123);

glVertex2f(269,130);

glVertex2f(264,128);

glVertex2f(270,100);

glVertex2f(274,104);

glVertex2f(272,90);

glVertex2f(275,130);

glVertex2f(265,127);

glVertex2f(255,133);

glEnd();

}

void steam()

{

//steam

for(i=0;i<21;i=i+1)

{

glColor4f(0.4,0.4,0.4,1.0);

glBegin(GL\_QUADS);

glVertex2f(120,185.5+i);

glVertex2f(120,189.5+i);

glColor4f(1.0,1.0,1.0,1.0);

glVertex2f(124,189.5+i);

glVertex2f(124,185.5+i);

glEnd();

glFlush();

delay(200);

}

for(i=0;i<117;i=i+1)

{

glBegin(GL\_QUADS);

glColor4f(1.0,1.0,1.0,1.0);

glVertex2f(124+i,210);

glVertex2f(128+i,210);

glColor4f(0.4,0.4,0.4,1.0);

glVertex2f(128+i,206);

glVertex2f(124+i,206);

glEnd();

glFlush();

delay(200);

}

for(i=0;i<33;i=i+1)

{

glBegin(GL\_QUADS);

glColor4f(1.0,1.0,1.0,1.0);

glVertex2f(240,206-i);

glColor4f(0.4,0.4,0.4,1.0);

glVertex2f(244,206-i);

glVertex2f(244,202-i);

glColor4f(1.0,1.0,1.0,1.0);

glVertex2f(240,202-i );

glEnd();

glFlush();

delay(200);

}

//glFlush();

glColor3f(0.4,0.4,0.4);

glBegin(GL\_QUADS);

glVertex2f(240,170);

glVertex2f(244,170);

glVertex2f(252,159);

glVertex2f(232,159);

glEnd();

glFlush();

}

void points1()

{

glPointSize(5.0);

glColor3f(0.0,0.0,1.0);

glBegin(GL\_POINTS);

glVertex2f(46,124);

glVertex2f(44,130);

glVertex2f(48,135);

glVertex2f(49,126);

glVertex2f(42,140);

glVertex2f(46,139);

glVertex2f(54,128);

glVertex2f(38,148);

glVertex2f(32,143);

glVertex2f(57,124);

glVertex2f(61,138);

glVertex2f(51,134);

glEnd();

}

void points2()

{

glPointSize(3.0);

glColor3f(0.4,0.4,0.4);

glBegin(GL\_POINTS);

glVertex2f(107,117);

glVertex2f(109,120);

glVertex2f(110,123);

glVertex2f(106,125);

glVertex2f(113,120);

glVertex2f(113,131);

glVertex2f(108,141);

glVertex2f(111,130);

glVertex2f(107,127);

glVertex2f(111,131);

glVertex2f(123,145);

glVertex2f(131,145);

glVertex2f(142,151);

glVertex2f(133,148);

glVertex2f(137,133);

glVertex2f(139,157);

glVertex2f(128,160);

glVertex2f(146,163);

glVertex2f(149,170);

glVertex2f(150,124);

glVertex2f(153,119);

glVertex2f(148,136);

glVertex2f(137,115);

glVertex2f(149,127);

glVertex2f(108,170);

glVertex2f(104,145);

glVertex2f(109,175);

glVertex2f(111,156);

glVertex2f(118,164);

glVertex2f(127,166);

glVertex2f(129,169);

glVertex2f(126,132);

glVertex2f(123,128);

glVertex2f(126,140);

glVertex2f(119,141);

glVertex2f(127,151);

glVertex2f(126,155);

glVertex2f(121,171);

glVertex2f(126,180);

glVertex2f(133,185);

glVertex2f(141,190);

glVertex2f(138,176);

glEnd();

}

void working()

{

//control rods

glClearColor(0,0,0,0);

glColor3f(0.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex3i(48,122,1);

glVertex3i(48,140,1);

glVertex3i(50,122,1);

glVertex3i(50,140,1);

glVertex3i(52,122,1);

glVertex3i(52,140,1);

glEnd();

glColor3f(0.6,0.6,0.4);

glBegin(GL\_LINES);

glVertex2i(48,140);

glVertex2i(48,185);

glVertex2i(50,140);

glVertex2i(50,185);

glVertex2i(52,140);

glVertex2i(52,185);

glEnd();

delay(500);

for(i=0,j=0;i<40,j<40;i=i+1,j=i+1)

{

glBegin(GL\_QUADS);

glColor3f(0.0,0.6,0.8);

glVertex2f(70+i,117);

glVertex2f(74+i,117);

glColor3f(1.0,1.0,1.0);

glVertex2f(74+i,121);

glVertex2f(70+i,121);

glEnd();

glBegin(GL\_QUADS);

glColor3f(0.0,0.6,0.8);

glVertex2f(350-j,89);

glVertex2f(280-j,89);

glColor3f(1.0,1.0,1.0);

glVertex2f(280-j,85);

glVertex2f(350-j,85);

glEnd();

glFlush();

delay(200);

}

for(i=0,j=0;i<32,j<25;i=i+1,j=j+1)

{

glBegin(GL\_QUADS);

glColor3f(1.0,1.0,1.0);

glVertex2f(111,117+i);

glColor3f(0.0,0.6,0.8);

glVertex2f(114,117+i);

glVertex2f(114,136+i);

glColor3f(1.0,1.0,1.0);

glVertex2f(111,136+i);

glEnd();

glBegin(GL\_QUADS);

glColor3f(0.0,0.6,0.8);

glVertex2f(240,89-j);

glColor3f(1.0,1.0,1.0);

glVertex2f(244,89-j);

glVertex2f(244,85-j);

glColor3f(0.0,0.6,0.8);

glVertex2f(240,85-j);

glEnd();

glFlush();

delay(200);

}

for(i=0,j=0;i<25,j<25;i=i+1,j=j+1)

{

glBegin(GL\_QUADS);

glColor3f(1.0,1.0,1.0);

glVertex2f(111+i,160);

glVertex2f(115+i,160);

glColor3f(0.0,0.6,0.8);

glVertex2f(115+i,163.5);

glVertex2f(111+i,163.5);

glEnd();

glBegin(GL\_QUADS);

glColor3f(1.0,1.0,1.0);

glVertex2f(244+j,65);

glVertex2f(326+j,65);

glColor3f(0.0,0.6,0.8);

glVertex2f(326+j,61);

glVertex2f(244+j,61);

glEnd();

glFlush();

delay(200);

}

for(i=0;i<112;i=i+1)

{

glBegin(GL\_QUADS);

glColor3f(1.0,1.0,1.0);

glVertex2f(135.2,160-i);

glColor3f(0.0,0.6,0.8);

glVertex2f(138.9,160-i);

glVertex2f(138.9,156-i);

glColor3f(1.0,1.0,1.0);

glVertex2f(135.2,156-i);

glEnd();

glFlush();

delay(200);

}

for(i=0;i<40;i=i+1)

{

glBegin(GL\_QUADS);

glColor3f(1.0,1.0,1.0);

glVertex2f(131-i,47.5);

glVertex2f(135.3-i,47.5);

glColor3f(0.0,0.6,0.8);

glVertex2f(135.3-i,44.5);

glVertex2f(131-i,44.5);

glEnd();

glFlush();

delay(200);

}

delay(500);

for(i=0;i<58;i=i+1)

{

glColor3f(1.0,1.0,1.0);

glBegin(GL\_QUADS);

glVertex2f(80-i,47.5);

glColor3f(0.0,0.6,0.8);

glVertex2f(80-i,44.5);

glVertex2f(76-i,44.5);

glColor3f(1.0,1.0,1.0);

glVertex2f(76-i,47.5);

glEnd();

glFlush();

delay(200);

}

for(i=0;i<92;i=i+1)

{

glBegin(GL\_QUADS);

glColor3f(0.0,0.6,0.8);

glVertex2f(16.5,47.5+i);

glColor3f(1.0,1.0,1.0);

glVertex2f(19.5,47.5+i);

glVertex2f(19.5,44.5+i);

glColor3f(0.0,0.6,0.8);

glVertex2f(16.5,44.5+i);

glEnd();

glFlush();

delay(200);

}

for(i=0;i<8;i=i+1)

{

glColor3f(0.0,0.6,0.8);

glBegin(GL\_QUADS);

glVertex2f(19.5+i,138.5);

glVertex2f(23+i,138.5);

glColor3f(1.0,1.0,1.0);

glVertex2f(23+i,135.5);

glVertex2f(19.5+i,135.5);

glEnd();

glFlush();

delay(200);

}

points2();

steam();

glPushMatrix();

while(p<200)

{

flag=1;

if(flag==1)

{

rot1();

}

/\*

if(flag==2)

{

rot2();

}

if(flag==3)

{

rot3();

}

if(flag==4)

{

rot4();

}

\*/

p++;

if(p==199)glEnable(GL\_DEPTH\_TEST);

}

glPopMatrix();

//delay(1000);

points();

conecting\_pipe();

}

void conecting\_pipe()

{

//coolant water

for(i=0;i<31;i=i+1)

{

glBegin(GL\_QUADS);

glColor3f(1.0,1.0,1.0);

glVertex2f(215-i,54);

glColor3f(0.0,0.6,0.8);

glVertex2f(215-i,50);

glVertex2f(225-i,50);

glColor3f(1.0,1.0,1.0);

glVertex2f(225-i,54);

glEnd();

glFlush();

delay(200);

}

for(i=0;i<43;i=i+1)

{

glBegin(GL\_QUADS);

glColor3f(0.0,0.6,0.8);

glVertex2f(185,54+i);

glColor3f(1.0,1.0,1.0);

glVertex2f(189,54+i);

glVertex2f(189,58+i);

glColor3f(0.0,0.6,0.8);

glVertex2f(185,58+i);

glEnd();

glFlush();

delay(200);

}

delay(500);

for(i=0;i<28;i=i+1)

{

glBegin(GL\_QUADS);

glColor3f(1.0,1.0,1.0);

glVertex2f(180-i,108);

glColor3f(0.0,0.6,0.8);

glVertex2f(180-i,104);

glVertex2f(176-i,104);

glColor3f(1.0,1.0,1.0);

glVertex2f(176-i,108);

glEnd();

glFlush();

delay(200);

}

}

void myMouse(int btn,int state,int x,int y);

void myMouse(int btn,int state,int x,int y)

{

}

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 setFont(void \*font)

{

currentfont=font;

}

void DrawCircle(float cx, float cy, float r, int num\_segments)

{

glBegin(GL\_LINE\_LOOP);

int ii;

for(ii = 0; ii < num\_segments; ii++)

{

float theta = 2.0f \* 3.1415926f \* ((float)(ii)) / ((float)(num\_segments));//get the current angle

float x = r \* cosf(theta);//calculate the x component

float y = r \* sinf(theta);//calculate the y component

if((y+cy)<cy)

break;

glVertex2f(x + cx, y + cy);//output vertex

}

glEnd();

}

void display\_nuclear\_power\_plant()

{

float PI = 22/7;

glLineWidth(6.0);

glClearColor(0.0,0.0,0.0,0.0);

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

//second set

int incrX = 10; //Change these to move the 3D box

int incrY = 20;

int copyX = incrX;

int copyY = incrY;

int baseX = 90;

int baseY = 200;

int someVal;

glLineWidth(20.0);

for(someVal=0;someVal<10;someVal++)

{

copyX = baseX + someVal;

copyY = baseY + (someVal \* 2);

DrawCircle(copyX,copyY,80,300);

}

glLineWidth(6.0);

/\*glBegin(GL\_LINE\_STRIP);

glColor3f(0.6,0.6,0.6);

glVertex2i(10,200);

glColor3f(0.1,0.3,0.0);

glVertex2i(10,30);

glVertex2i(170,30);

glColor3f(0.6,0.6,0.6);

glVertex2i(170,200);

glEnd();\*/

//BACK WALL OF nuclear reactor

glColor3f(0.6,0.6,0.6);

glBegin(GL\_LINE\_STRIP);

glVertex2i(10+incrX,30+incrY); //bottom left

glVertex2i(170+incrX,30+incrY); //bottom right

glVertex2i(170+incrX,200+incrY); //top right

glVertex2i(10+incrX,200+incrY); //top left

glEnd();

//Base of nuclear reactor 3D

glBegin(GL\_LINE\_STRIP);

glVertex2i(10,30); //bottom left

glVertex2i(170,30); //bottom right

glVertex2i(170+incrX,30+incrY); //top right

glVertex2i(10+incrX,30+incrY); //top left

glEnd();

//Right wall of nuclear reactor 3D

glBegin(GL\_LINE\_STRIP);

glVertex2i(170,30); //bottom left

glVertex2i(170+incrX,30+incrY); //bottom right

glVertex2i(170+incrX,200+incrY); //top right

glVertex2i(170,200); //top left

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2i(170,30); //bottom left

glVertex2i(170,200); //top left

glEnd();

//Left wall of nuclear reactor 3D

glBegin(GL\_LINE\_STRIP);

glVertex2i(10,30); //bottom left

glVertex2i(10+incrX,30+incrY); //bottom right

glVertex2i(10+incrX,200+incrY); //top right

glVertex2i(10,200); //top left

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2i(10,30); //bottom left

glVertex2i(10,200); //top left

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2i(10,200); //bottom left

glVertex2i(170,200); //top left

glEnd();

/\*glLineWidth(3.0);

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0,0.0,0.0);

glVertex2i(10,30);

glVertex2i(10+incrX,30+incrY);

glVertex2i(10+incrX,200+incrY);

glVertex2i(10,200);

glVertex2i(10,30);

glVertex2i(170,30);

glVertex2i(170+incrX,30+incrY);

glVertex2i(170+incrX,200+incrY);

glVertex2i(170,200);

glVertex2i(170,30);

glVertex2i(170+incrX,30+incrY);

glVertex2i(10+incrX,30+incrY);

glEnd();\*/

//draw semicircle

/\*

int progY=220,r11;

for(r11=80;r11<90;r11++)

{

glColor3f(0.6,0.6,0.6);

glBegin(GL\_LINE\_STRIP);

for(angle=0; angle<=180; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 90 + r11\* (float)cos(angle\_radians);

y = progY + r11\* (float)sin(angle\_radians);

glLineWidth(6.0);

glVertex2f(x,y);

}

glEnd();

}

r11 = 80;

for(progY=200;progY<260;progY++)

{

}\*/

/\*glColor3f(0.6,0.6,0.6);

glBegin(GL\_LINE\_STRIP);

for(angle=0; angle<=180; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 90 + r1\* (float)cos(angle\_radians);

y = 200 + r1\* (float)sin(angle\_radians);

glLineWidth(6.0);

glVertex2f(x,y);

}

glEnd();\*/

//draw 1st inner container

glColor3f(1.0,1.0,1.0);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(42.0,98.0,0.0,"core");

glColor3f(0.6,0.6,0.6);

glBegin(GL\_LINE\_LOOP);

glVertex2i(25,150);

glVertex2i(25,75);

glVertex2i(75,75);

glVertex2i(75,150);

glEnd();

glColor3f(0.6,0.6,0.6);

glBegin(GL\_LINES);

glVertex2i(100,100);

glVertex2i(100,170);

glVertex2i(155,100);

glVertex2i(155,170);

glEnd();

glBegin(GL\_LINE\_STRIP);

for(angle=0; angle<=180; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 127.5 + r3\* (float)cos(angle\_radians);

y = 170 + r3\* (float)sin(angle\_radians);

glLineWidth(6.0);

glVertex2f(x,y);

}

glEnd();

glBegin(GL\_LINE\_STRIP);

for(angle=180; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 127.5 + r3\* (float)cos(angle\_radians);

y = 100 + r3\* (float)sin(angle\_radians);

glLineWidth(6.0);

glVertex2f(x,y);

}

glEnd();

//core container

glLineWidth(5.0);

glColor3f(0.6,0.6,0.6);

glBegin(GL\_LINE\_LOOP);

glVertex2i(42,105);

glVertex2i(57,105);

glVertex2i(57,120);

glVertex2i(42,120);

glEnd();

//draw core

glBegin(GL\_POLYGON);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 49 + r2\* (float)cos(angle\_radians);

y = 112.5 + r2\* (float)sin(angle\_radians);

glColor3f(1.0,0.4,0.2);

glLineWidth(6.0);

glVertex2f(x,y);

}

glEnd();

//2nd outer container

glLineWidth(6.0);

glColor3f(0.6,0.6,0.6);

glBegin(GL\_LINE\_LOOP);

glVertex2i(220,170);

glColor3f(0.1,0.3,0.0);

glVertex2i(220,30);

glVertex2i(330,30);

glColor3f(0.6,0.6,0.6);

glVertex2i(330,220);

glEnd();

//pipe frm cont1

glLineWidth(3.0);

glColor3f(0.6,0.3,0.2);

glBegin(GL\_LINE\_STRIP);

glVertex2i(30,135);

glVertex2i(20,135);

glVertex2i(20,48);

glVertex2i(80,48);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2i(80,44);

glVertex2i(16,44);

glVertex2i(16,139);

glVertex2i(30,139);

glEnd();

//pipe frm con2

glColor3f(0.6,0.3,0.2);

glBegin(GL\_LINE\_STRIP);

glVertex2i(70,117);

glVertex2i(115,117);

glVertex2i(115,160);

glVertex2i(135,160);

glVertex2i(135,48);

glVertex2i(92,48);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2i(92,44);

glVertex2i(139,44);

glVertex2i(139,164);

glVertex2i(111,164);

glVertex2i(111,121);

glVertex2i(70,121);

glEnd();

glColor3f(0.6,0.3,0.2);

//pump1

glBegin(GL\_POLYGON);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 86 + r4\* (float)cos(angle\_radians);

y = 46 + r4\* (float)sin(angle\_radians);

glVertex2f(x,y);

}

glEnd();

glColor4f(0.0,0.0,0.0,1.0);

glLineWidth(6.0);

glBegin(GL\_LINES);

glVertex2i(80,48);

glVertex2i(92,44);

glVertex2i(80,44);

glVertex2i(92,48);

glEnd();

//upper pipe

glLineWidth(3.0);

glColor3f(0.6,0.3,0.2);

glBegin(GL\_LINES);

glVertex2i(240,170);

glVertex2f(240,178);

glVertex2f(240,180);

glVertex2i(240,206);

glVertex2i(240,206);

glVertex2i(124,206);

glVertex2i(124,206);

glVertex2f(124,198);

glVertex2f(124,196);

glVertex2i(124,185);

glEnd();

glBegin(GL\_LINES);

glVertex2i(120,185);

glVertex2i(120,195);

glVertex2i(120,197);

glVertex2i(120,210);

glVertex2i(120,210);

glVertex2i(244,210);

glVertex2i(244,210);

glVertex2i(244,182);

glVertex2i(244,180);

glVertex2i(244,170);

glEnd();

//coolant pipe

glBegin(GL\_LINES);

glVertex2i(350,85);

glVertex2f(331,85);

glVertex2f(329,85);

glVertex2i(244,85);

glVertex2i(244,85);

glVertex2i(244,65);

glVertex2i(244,65);

glVertex2i(329,65);

glVertex2i(331,65);

glVertex2i(350,65);

glEnd();

glBegin(GL\_LINES);

glVertex2i(350,89);

glVertex2i(331,89);

glVertex2i(329,89);

glVertex2i(240,89);

glVertex2i(240,89);

glVertex2i(240,61);

glVertex2i(240,61);

glVertex2i(329,61);

glVertex2i(331,61);

glVertex2i(350,61);

glEnd();

//control rods

glColor3f(0.6,0.6,0.4);

glBegin(GL\_LINES);

glVertex2i(48,120);

glVertex2i(48,165);

glVertex2i(50,120);

glVertex2i(50,165);

glVertex2i(52,120);

glVertex2i(52,165);

glEnd();

glColor3f(0.3,0.1,0.1);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(42.0,98.0,0.0,"core");

turbine();

generator();

connecting\_pipe();

//pump2

glColor3f(0.6,0.3,0.2);

glBegin(GL\_POLYGON);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 186 + r5\* (float)cos(angle\_radians);

y = 106 + r5\* (float)sin(angle\_radians);

glLineWidth(6.0);

glVertex2f(x,y);

}

glEnd();

/\*glBegin(GL\_LINES);

glVertex2f(181,109);

glVertex2f(191.5,102);

glVertex2f(181,101.5);

glVertex2f(190,109.5 );

glEnd();

\*/

glColor3f(1.0,1.0,1.0);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(75.0,230.0,0.0,"constraint structure");

glColor3f(1.0,1.0,1.0);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(54.0,160.0,0.0,"control rods");

glColor3f(1.0,1.0,1.0);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(30.0,60.0,0.0,"reactor vessel");

glColor3f(1.0,1.0,1.0);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(250.0,105.0,0.0,"turbine");

glColor3f(1.0,1.0,1.0);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(180.0,120.0,0.0,"pump");

glColor3f(1.0,1.0,1.0);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(180.0,215.0,0.0,"steam line");

glColor3f(1.0,1.0,1.0);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(102.0,170.0,0.0,"steam generator");

glColor3f(1.0,1.0,1.0);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(80.0,55.0,0.0,"pump");

glColor3f(1.0,1.0,1.0);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(290.0,150.0,0.0,"generator");

glColor3f(1.0,1.0,1.0);

setFont(GLUT\_BITMAP\_HELVETICA\_18);

drawstring(248.0,70.0,0.0,"condensor cooling water");

//blended water

glBegin(GL\_POLYGON);

glColor3f(0.0,0.0,1.0);

glVertex2f(26,122);

glColor3f(0.0,0.6,0.8);

glVertex2f(26,76);

glVertex2f(74,76);

glColor3f(0.0,0.0,0.5);

glVertex2f(74,122);

glEnd();

//blended container

glColor3f(0.0,0.6,0.8);

glBegin(GL\_POLYGON);

glVertex2i(100,100);

glVertex2i(155,100);

glColor3f(0.0,0.0,1.0);

glVertex2i(155,113);

glVertex2i(100,113);

glEnd();

glColor3f(0.0,0.6,0.8);

glBegin(GL\_POLYGON);

for(angle=180; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 127.5 + r3\* (float)cos(angle\_radians);

y = 100 + r3\* (float)sin(angle\_radians);

glLineWidth(6.0);

glVertex2f(x,y);

}

glEnd();

//glFlush();

//water in cont2

glBegin(GL\_POLYGON);

glColor3f(0.0,0.6,0.8);

glVertex2f(221,31);

glVertex2f(329,31);

glColor3f(0.0,0.0,1.0);

glVertex2f(329,57);

glVertex2f(221,57);

glEnd();

glFlush();

delay(1000);

points1();

delay(1000);

working();

glFlush();

}

void display\_about(void)

{

glClearColor(0.7,0.3,0.1,1.0);

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,1.0,1.0);

drawstring(140.0,275.0,0.0,"DEPARTMENT OF COMPUTER SCIENCE");

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,1.0,1.0);

drawstring(105,245,0.0,"CG MINI PROJECT ON WORKING OF NUCLEAR POWER PLANT");

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,1.0,1.0);

drawstring(190,230,0,"USING OPEN GL");

glColor3f(0.0,1.0,1.0);

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,1.0,1.0);

glColor3f(0.0,0.0,0.0);

drawstring(300,15,0.0,"RIGHT CLICK for Menu");

glFlush();

}

void reactions(void)

{

glClearColor(0.0,0.0,0.0,0.0);

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

//glDisable(GL\_DEPTH\_TEST);

glPointSize(10.0);

glFlush();

glBegin(GL\_POLYGON);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 200 + r8\* (float)cos(angle\_radians);

y = 270 + r8\* (float)sin(angle\_radians);

glColor3f(0.8,0.3,0.2);

glVertex2f(x,y);

}

glEnd();

glBegin(GL\_POLYGON);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 200 + r8\* (float)cos(angle\_radians);

y = 92 + r8\* (float)sin(angle\_radians);

glColor3f(0.8,0.3,0.2);

glVertex2f(x,y);

}

glEnd();

glBegin(GL\_POLYGON);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 178 + r8\* (float)cos(angle\_radians);

y = 114 + r8\* (float)sin(angle\_radians);

glColor3f(0.8,0.3,0.2);

glVertex2f(x,y);

}

glEnd();

glBegin(GL\_POLYGON);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 222 + r8\* (float)cos(angle\_radians);

y = 114 + r8\* (float)sin(angle\_radians);

glColor3f(0.8,0.3,0.2);

glVertex2f(x,y);

}

glEnd();

glColor3f(0.0,1.0,0.6);

glLineWidth(6.0);

glBegin(GL\_LINES);

glVertex2f(200.0,267.0);

glVertex2f(200.0,201.0);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2f(198.0,204.0);

glVertex2f(200.0,198.0);

glVertex2f(202.0,204.0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(91.0,180.0);

glVertex2f(182.0,180.0);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2f(94.0,183.0);

glVertex2f(91.0,180.0);

glVertex2f(94.0,177.0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(218.0,180.0);

glVertex2f(306.0,180.0);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2f(302.0,183.0);

glVertex2f(306.0,180.0);

glVertex2f(302.0,177.0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(200.0,162.0);

glVertex2f(200.0,97.0);

glVertex2f(200.0,162.0);

glVertex2f(220.0,117.0);

glVertex2f(200.0,162.0);

glVertex2f(180.0,117.0);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2f(197.0,101.0);

glVertex2f(200.0,96.0);

glVertex2f(203.0,101.0);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2f(177.0,119.0);

glVertex2f(180.0,117.0);

glVertex2f(183.0,117.0);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2f(217.0,117.0);

glVertex2f(220.0,117.0);

glVertex2f(223.0,119.0);

glEnd();

glBegin(GL\_POLYGON);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 200 + r10\* (float)cos(angle\_radians);

y = 180 + r10\* (float)sin(angle\_radians);

glColor3f(0.6,0.1,0.9);

glVertex2f(x,y);

}

glEnd();

/\*glColor3f(0.6,0.1,0.9);

glBegin(GL\_POLYGON);

glVertex2f(242,155);

glVertex2f(198,198);

glVertex2f(183,189);

glEnd();

\*/for(r=17;r>=0;r--)

{

glBegin(GL\_LINE\_LOOP);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 200 + r\* (float)cos(angle\_radians);

y = 180 + r\* (float)sin(angle\_radians);

glColor3f(0.6,0.1,0.9);

glVertex2f(x,y);

}

glEnd();

}

glBegin(GL\_POLYGON);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 320 + r13\* (float)cos(angle\_radians);

y = 180 + r13\* (float)sin(angle\_radians);

glColor4f(1.0,0.0,0.0,0.6);

glVertex2f(x,y);

}

glEnd();

for(r13=13;r13>=0;r13--)

{

glBegin(GL\_LINE\_LOOP);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 320 + r13\* (float)cos(angle\_radians);

y = 180 + r13\* (float)sin(angle\_radians);

glColor4f(1.0,0.0,0.0,0.6);

glVertex2f(x,y);

}

glEnd();

}

glBegin(GL\_POLYGON);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 80 + r11\* (float)cos(angle\_radians);

y = 180 + r11\* (float)sin(angle\_radians);

glColor4f(0.0,1.0,0.0,0.5);

glVertex2f(x,y);

}

glEnd();

for(r11=9;r11>=0;r11--)

{

glBegin(GL\_LINE\_LOOP);

for(angle=0; angle<=360; angle=angle+5)

{

angle\_radians = angle \* (float)3.14159 / (float)180;

x = 80 + r11\* (float)cos(angle\_radians);

y = 180 + r11\* (float)sin(angle\_radians);

glColor4f(0.0,1.0,0.0,0.5);

glVertex2f(x,y);

}

glEnd();

}

setFont(GLUT\_BITMAP\_HELVETICA\_18);

glColor3f(1.0,1.0,1.0);

drawstring(210,160,0.0,"U235");

setFont(GLUT\_BITMAP\_HELVETICA\_18);

glColor3f(1.0,1.0,1.0);

drawstring(190,290,0.0,"neutron");

setFont(GLUT\_BITMAP\_HELVETICA\_18);

glColor3f(1.0,1.0,1.0);

drawstring(190,280,0.0,"(thermal)");

setFont(GLUT\_BITMAP\_HELVETICA\_18);

glColor3f(1.0,1.0,1.0);

drawstring(70,155,0.0,"Fission product1");

setFont(GLUT\_BITMAP\_HELVETICA\_18);

glColor3f(1.0,1.0,1.0);

drawstring(70,145,0.0,"Light Nucleus");

setFont(GLUT\_BITMAP\_HELVETICA\_18);

glColor3f(1.0,1.0,1.0);

drawstring(306,155,0.0,"Fission product2");

setFont(GLUT\_BITMAP\_HELVETICA\_18);

glColor3f(1.0,1.0,1.0);

drawstring(306,145,0.0,"Heavy Nucleus");

setFont(GLUT\_BITMAP\_HELVETICA\_18);

glColor3f(1.0,1.0,1.0);

drawstring(165,80,0.0,"About 3 neutrons per fission");

glColor3f(1.0,1.0,1.0);

glPointSize(10.0);

glBegin(GL\_POINTS);

glVertex2f(10,52);

glEnd();

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(1.0,1.0,1.0);

drawstring(20,50,0.0,"neutrons produced can initiate further fission of U235, making the reaction to be");

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(1.0,1.0,1.0);

drawstring(20,40,0,"self sustainable chain reaction");

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,0.0,1.0);

drawstring(130,20,0,"NUCLEAR FISSION REACTION");

glFlush();

}

/\*void key(unsigned char key,int x,int y)

{

if(key==13 && enter==1)

{

reactions();

}

}\*/

void display\_operations(void)

{

glClearColor(0.6,0.6,0.0,0.0);

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2i(80,230);

glVertex2i(82,230);

glVertex2i(82,232);

glVertex2i(80,232);

glEnd();

drawstring(90.0,230.0,0.0,"Nuclear powerplant is a power generating unit in which nuclear energy\n");

glColor3f(0.0,0.0,0.0);

drawstring(90.0,220.0,0.0,"is converted to electrical energy.");

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2i(80,200);

glVertex2i(82,200);

glVertex2i(82,202);

glVertex2i(80,202);

glEnd();

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,0.0,0.0);

drawstring(90.0,200.0,0.0,"Turbo Generator made to rotate from heat generated by nuclear fission ");

glColor3f(0.0,0.0,0.0);

drawstring(90,190,0.0,"reactions of radio active elements such as Uranium-235,plutonium");

glColor3f(0.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2i(80,170);

glVertex2i(82,170);

glVertex2i(82,172);

glVertex2i(80,172);

glEnd();

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,0.0,0.0);

drawstring(90,170,0.0,"Radioactive elements splits when bombarded by neutron.Reactions become");

glColor3f(0.0,0.0,0.0);

drawstring(90,160,0.0,"self sustaining chain reaction under controlled conditions.");

glColor3f(0.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2i(80,140);

glVertex2i(82,140);

glVertex2i(82,142);

glVertex2i(80,142);

glEnd();

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,0.0,0.0);

drawstring(90,140,0.0,"Heat Energy generated is transfered to ordinary water and carried away");

glColor3f(0.0,0.0,0.0);

drawstring(90,130,0.0,"from reactor as steam.");

glColor3f(0.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2i(80,110);

glVertex2i(82,110);

glVertex2i(82,112);

glVertex2i(80,112);

glEnd();

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,0.0,0.0);

drawstring(90,110,0.0,"Steem feeds the Turbo Generator thus making it to rotate and");

glColor3f(0.0,0.0,0.0);

drawstring(90,100,0.0,"power is generated.");

/\*setFont(GLUT\_BITMAP\_HELVETICA\_18);

glColor3f(0.0,0.0,0.0);

drawstring(250,15,0.0,"Press ENTER to See Internal Chain Reaction");

\*/

delay(1000);

//reactions();

glFlush();

}

/\*

void ext()

{

glClearColor(0.0,0.4,0.6,0.0);

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,0.0,0.0);

drawstring(100,110,0.0,"THANK YOU");

delay(2000);

}

\*/

void options(int id)

{

int in\_about;

switch(id)

{

case 1:

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

in\_about=1;

display\_about();

break;

case 2:

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

// glColor3f(0.6,0.3,0.0);

display\_nuclear\_power\_plant();

break;

case 3:

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

display\_operations();

break;

case 4:glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

glColor3f(0.6,0.3,0.0);

reactions();

break;

case 5:

exit(0);

}

}

void disp(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

display\_about();

glFlush();

// glutSwapBuffers();

//glutPostRedisplay();

}

void init(void)

{

glClearColor(0.0,0.0,0.0,0.0);

glEnable(GL\_DEPTH\_TEST);

gluOrtho2D(0.0,400.0,0.0,300.0);

// glPointSize(6.0);

// glLineWidth(4.0);

// glMatrixMode(GL\_PROJECTION);

// glLoadIdentity();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGBA);

glutInitWindowPosition(50,50);

glutInitWindowSize(400,300);

glutCreateWindow("nuclear plant");

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA,GL\_ONE\_MINUS\_SRC\_ALPHA);

init();

enter=1;

glutCreateMenu(options);

glutAddMenuEntry("About the Project",1);

glutAddMenuEntry("working of power plant",2);

glutAddMenuEntry("About nuclear power plant",3);

glutAddMenuEntry("internal reactions",4);

glutAddMenuEntry("Quit",5);

glutMouseFunc(myMouse);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

// glutAttachMenu(GLUT\_LEFT\_BUTTON);

// glutReshapeFunc(myReshape);

glutDisplayFunc(disp);

// glutKeyboardFunc(key);

glutMainLoop();

}

**CHAPTER 6**

**6. Function Description**

The description of all the functions used in the program is given below:

***Void glutInit (int \*argc, char \*\*argv)***

* This function initializes GLUT. The arguments from main are passed in and can beused by application.

***Void glutInitDisplayMode (unsigned int mode)***

* This function requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model (GLUT\_RGB, GLUT\_INDEX) and buffering (GLUT\_SINGLE, GLUT\_DOUBLE).

***Void glutInitWindowPosition (int x, int y)***

* This specifies the initial position of top-left corner of the windows in pixels.

***Void glutInitWindowSize (int width, int height)***

* This function specifies the initial height and width of the window in pixels.

***Void glutCreateWindow (char \*title)***

* This function creates a window on the display the string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

***Void glutDisplayFunc (void (\*func) (void))***

* This function registers the display func that is executed when the window needs to be redrawn.

***Void glClearColor(GLclampfr,GLclampf g, GLclampfb,GLclampf a)***

* This sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating point numbers between 0.0 and 1.0.

***Void glClear(GLbitfield mask)***

* It clear buffers to present values. The value of mask is determined by the bitwise OR of options GL\_COLOR\_BUFFER\_BIT, GL\_DEPTH\_BUFFER\_BIT

***Void glutPostRedisplay ()***

* This function requests that the display callback be executed after the current callback returns.

***Void glutReshapeFunc (void \*f (int width, int height)***

* This function registers the reshape callback function f. The callback function returns the height and width of the new window. The reshape callback invokes a display callback.

***Void glViewport (int x, int y, GLsizei width, GLsizei height)***

* This function specifies a width\*height viewport in pixels whose lower left corner is at (x, y) measured from the origin of the window.

***Void glMatrixMode (GLenum mode)***

* This function specifies which matrix will be affected by subsequent transformations. Mode can be GL\_MODEL\_VIEW, GL\_PROJECTION, GL\_TEXTURE.

***Void glLoadIdentity ()***

* This function sets the current transformation matrix to an identity matrix.

***Void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)***

* It defines the orthographic viewing volume with all parameters measured from the center of the projection plane

***Void glutKeyboardFunc(void \*f(char key,intwidth,int height))***

* This function registers the keyboard callback function f. the callback function returns the ASCII code of the key pressed and the position of the mouse.

***Void glutMouseFunc (void \*f (int button, int state, int x, int y)***

* This function registers the mouse callback function f. The callback function returns the button(GLUT\_LEFT\_BUTTON,GLUT\_MIDDLE\_BUTTON,GLUT\_RIGHT\_BUTTON), the state of the button after the event (GLUT\_UP, GLUT\_DOWN), and the position of the mouse relative to the top-left corner of the window.

***intglutCreateMenu (void (\*f) (int value))***

* This function returns for a top-level menu and registers the callback function that returns an integer value corresponding to the menu entry selected.

***Void glutAddMenuEntry (char \*name, int value)***

* This function adds an entry with the string name displayed to the current menu. value is returned to the menu callback when the entry is selected.

***Void glut AttachMenu(int button)***

* This function attachés the current menu to the specified mouse button.

***Void glRasterpos2d(TYPE x-coordinate, TYPE y-coordinate)***

* This specifies the raster position

***Void glutBitmapCharacter(void \*font, int char)***

* This renders the character with ASCII code char at the current raster position using the raster font given by font. Fonts include GLUT\_BITMAP\_TIMES\_ROMAN\_10 and GLUT\_BITMAP\_TIMES\_ROMAN\_8\_BY\_13. The raster position is incremented by the width of the character

***Void glVertex3f(TYPE x-coordinate, TYPE y-coordinate, TYPE z-coordinate)***

***Void glVertex3fv(TYPE \*coordinates)***

* This specifies the position of a vertex in 3 dimension. If v is present, the argument is a pointer to an array containing the coordinates.

***Void glColor3f (TYPE r, TYPE g, TYPE b)***

***Void glColor3fv(TYPE \*color)***

* This function sets the present RGB colors. The maximum and minimum values of floating point types are 1.0 and 0.0. If v is present, the argument is a pointer to an array containing the colors.

***Void glBegin(glEnum mode)***

* This initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL\_POINTS, GL\_LINES AND GL\_POLYGON.

***Void glEnd()***

* It terminates a list of vertices.

***Void glFlush()***

* This forces any buffered OpenGL commands to execute.

***Void glutMainLoop ()***

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

**CHAPTER 7**

**7. Snapshots**

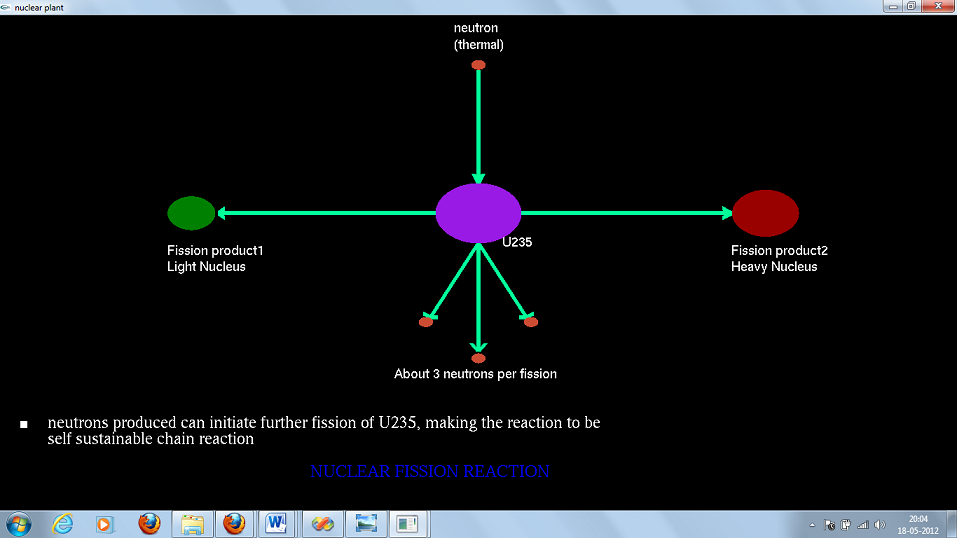


Fig 6.1 Nuclear Fissoin Reaction

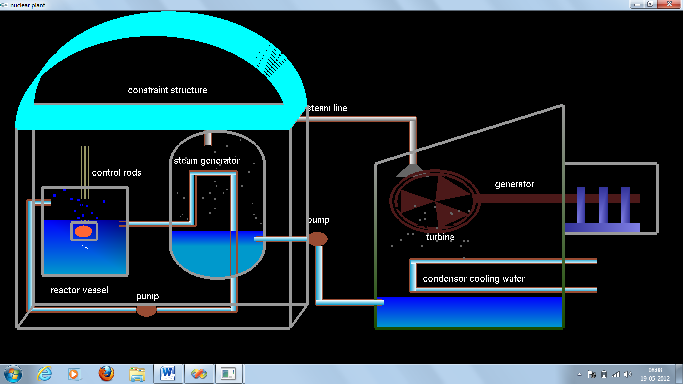


Fig 6.2 Working of nuclear Plant

**8. Conclusion**

The aim of the project is to demonstrate the working of nuclear plant. The code has been written in openGL. The code contains the functions that accomplish all the tasks required for the project. It can be further improved upon to provide better facilities and user interface.

The development of this project was very helpful in developing our programming skills. The project helped us to understand COMPUTER GRAPHICS and OPEN GL to greater depth. We found designing and developing this project interesting and a good learning experience.The experienced gained during the course of development of this project will help in future endeavors.

We conclude the project ‘WORKING OF NUCLEAR PLANT’ successfully truest of our senses and to best of our ability.