Text

Description automatically generated with low confidence

**LAB MANUAL**

**Computer Graphics and Visualization Laboratory with Mini Project**

**[18CSL67]**

Mrs.Tessy Abraham

Assistant Professor Department of CSE

**TABLE OF CONTENTS**

**Part A**

|  |  |
| --- | --- |
| **SL NO** | **PROGRAMS** |
| 01 | Implement Brenham’s line drawing algorithm for all types of slope |
| 02 | Create and rotate a triangle about the origin and a fixed point |
| 03 | Draw a color cube and spin it using OpenGL transformation matrices |
| 04 | Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing |
| 05 | Clip a lines using Cohen-Sutherland algorithm |
| 06 | To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene |
| 07 | Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user |
| 08 | Develop a menu driven program to animate a flag using Bezier Curve algorithm |
| 09 | Develop a menu driven program to fill the polygon using scan line algorithm |
|  | VIVA QUESTIONS |

**Introduction to Computer Graphics**

**Introduction to Open GL**

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. With OpenGL, you can build up your desired model from a small set of *geometric primitives* - points, lines, and polygons. A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features. GLU is a standard part of every OpenGL implementation.

**OpenGL as a State Machine**

OpenGL is a state machine. It is called a state machine because it can be put into various states until you change them. As you've already seen, the current color is a state variable. You can set the current color to white, red, or any other color, and thereafter every object is drawn with that color until you set the current color to something else.

The current color is only one of many state variables that OpenGL maintains. Others control such things as the current viewing and projection transformations; line and polygon stipple patterns, polygon drawing modes, pixel-packing conventions, positions and characteristics of lights, and material properties of the objects being drawn. Many state variables refer to modes that are enabled or disabled with the command **glEnable()** or **glDisable()**. Each state variable or mode has a default value, and at any point you can query the system for each variable's current value.

## OpenGL-Related Libraries

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to allow you to simplify your programming tasks, including the following:

* The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation. GLU routines use the prefix **glu**.
* The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit. It contains rendering commands but is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse. Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), GLUT includes several routines that create more complicated three-dimensional objects such as a sphere, a torus, and a teapot. GLUT may not be satisfactory for full-featured OpenGL applications, but you may find it a useful starting point for learning OpenGL.

### Include Files

For all OpenGL applications, you want to include the gl.h header file in every file. Almost all OpenGL applications use GLU, the aforementioned OpenGL Utility Library, which requires inclusion of the glu.h header file. So almost every OpenGL source file begins with

#include <GL/gl.h>

#include <GL/glu.h>

If you are using GLUT for managing your window manager tasks, you should include

#include <GL/glut.h>

Note that glut.h includes gl.h, glu.h automatically, so including all three files is redundant.

## Associated utility libraries

Several libraries are built on top of or beside OpenGL to provide features not available in OpenGL itself. Libraries such as [GLU](http://en.wikipedia.org/wiki/OpenGL_Utility_Library) can be found with most OpenGL implementations, and others such as [GLUT](http://en.wikipedia.org/wiki/OpenGL_Utility_Toolkit) and [SDL](http://en.wikipedia.org/wiki/Simple_DirectMedia_Layer) have grown over time and provide rudimentary cross-platform windowing and mouse functionality, and if unavailable can easily be downloaded and added to a development environment. Simple [graphical user interface](http://en.wikipedia.org/wiki/Graphical_user_interface) functionality can be found in libraries like [GLUI](http://en.wikipedia.org/wiki/OpenGL_User_Interface_Library) or [FLTK](http://en.wikipedia.org/wiki/FLTK). Still other libraries like GLAux (OpenGL Auxiliary Library) are deprecated and have been superseded by functionality commonly available in more popular libraries.

Other libraries have been created to provide OpenGL application developers a simple means of managing OpenGL extensions and versioning. Examples of these libraries include [GLEW](http://en.wikipedia.org/wiki/OpenGL_Extension_Wrangler_Library) (the OpenGL Extension Wrangler Library) and [GLEE](http://en.wikipedia.org/wiki/OpenGL_Easy_Extension_library) (the OpenGL Easy Extension Library). In addition to the aforementioned simple libraries, other higher-level object-oriented scene graph [retainedmode](http://en.wikipedia.org/wiki/Retained_mode" \o "Retained mode) libraries exist such as [PLIB](http://en.wikipedia.org/wiki/PLIB), [OpenSG](http://en.wikipedia.org/wiki/OpenSG" \o "OpenSG), [OpenSceneGraph](http://en.wikipedia.org/wiki/OpenSceneGraph" \o "OpenSceneGraph), and [OpenGL Performer](http://en.wikipedia.org/wiki/OpenGL_Performer). These are available as cross-platform free/open source or [proprietary](http://en.wikipedia.org/wiki/Proprietary_software) programming interfaces written on top of OpenGL and systems libraries to enable the creation of [real-time](http://en.wikipedia.org/wiki/Real-time_computing) visual simulation applications.

Comprises several libraries with varying levels of abstraction: GL, GLU, and GLUT

* Software Interface to Graphics Hardware
* Consists of about 150 Distinct Commands
* Hardware-independent Interface
  + no command for windows or user input handling
  + does not include low-level I/O management
* Mid-level, device-independent, portable graphics subroutine package
* Developed primarily by SGI
* 2D/3D graphics, lower-level primitives (polygons)
* Basis for higher-level libraries/toolkits

**OpenGL Hierarchy**

* **Several levels of abstraction are provided**
* **GL** 
  + Lowest level: vertex, matrix manipulation
  + glVertex3f(point.x, point.y, point.z)
* **GLU**
  + Helper functions for shapes, transformations
  + gluPerspective( fovy, aspect, near, far )
  + gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
* **GLUT**
  + Highest level: Window and interface management
  + glutSwapBuffers()
  + glutInitWindowSize (500, 500);

**OpenGL Implementations**

* OpenGL IS an API (think of as collection of .h files):
  + #include <GL/gl.h>
  + #include <GL/glu.h>
  + #include <GL/glut.h>
* Windows, Linux, UNIX, etc. all provide a platform specific implementation.
* Windows: opengl32.lib glu32.lib glut32.lib
* Linux: -l GL -l GLU –l GLUT

**OpenGL API**

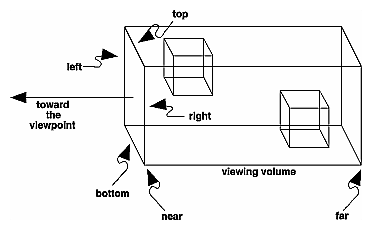
* As a programmer, you need to do the following things:
  + Specify the location/parameters of camera.
  + Specify the geometry (and appearance).
  + Specify the lights (optional).
* OpenGL will compute the resulting 2D image

**OpenGL: Camera**

* Two things to specify:
  + Physical location of camera in the scene (MODELVIEW matrix in OpenGL).
  + Projection properties of the camera (PROJECTION matrix in OpenGL):

****

void glFrustum(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far);

****

void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,

GLdouble top, GLdouble near, GLdouble far);

**OpenGL Conventions**

* Many functions have multiple forms:
  + glVertex2f, glVertex3i, glVertex4dv, etc.
* Number indicates number of arguments
* Letters indicate type
  + f: float, d: double, ub: unsigned byte, etc.
* ‘v’ (if present) indicates a single pointer argument

**Required files for Windows**

* In the System Directory
  + glu32.dll
  + opengl32.dll
  + glut32.dll
* In the C++ Include Directory
  + gl\gl.h
  + l\glu.h
  + gl\glaux.h (probably won't need it)
  + gl\glut.h (includes both gl.h and glu.h)
* In the C++ Library Directory
  + gl\glu32.lib
  + l\opengl32.lib
  + gl\glaux.lib (probably won't need it)
  + gl\glut32.lib

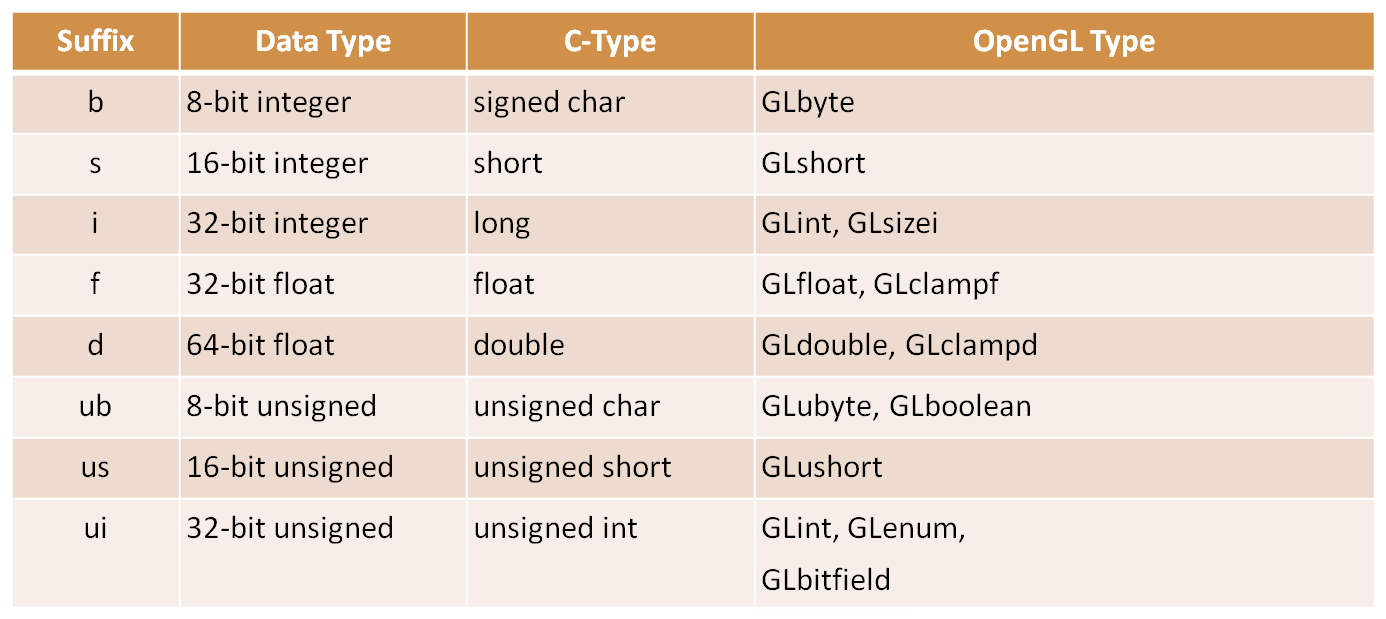
**Event Loop**

* OpenGL programs often run in an event loop:
  + Start the program
  + Run some initialization code
  + Run an infinite loop and wait for events such as
    - Key press
    - Mouse move, click
    - Reshape window
    - Expose event

**OpenGL Command Syntax (1)**

* OpenGL commands start with “gl”
* OpenGL constants start with “GL\_”
* Some commands end in a number and one, two or three letters at the end (indicating number and type of arguments)
* A Number indicates number of arguments
* Characters indicate type of argument

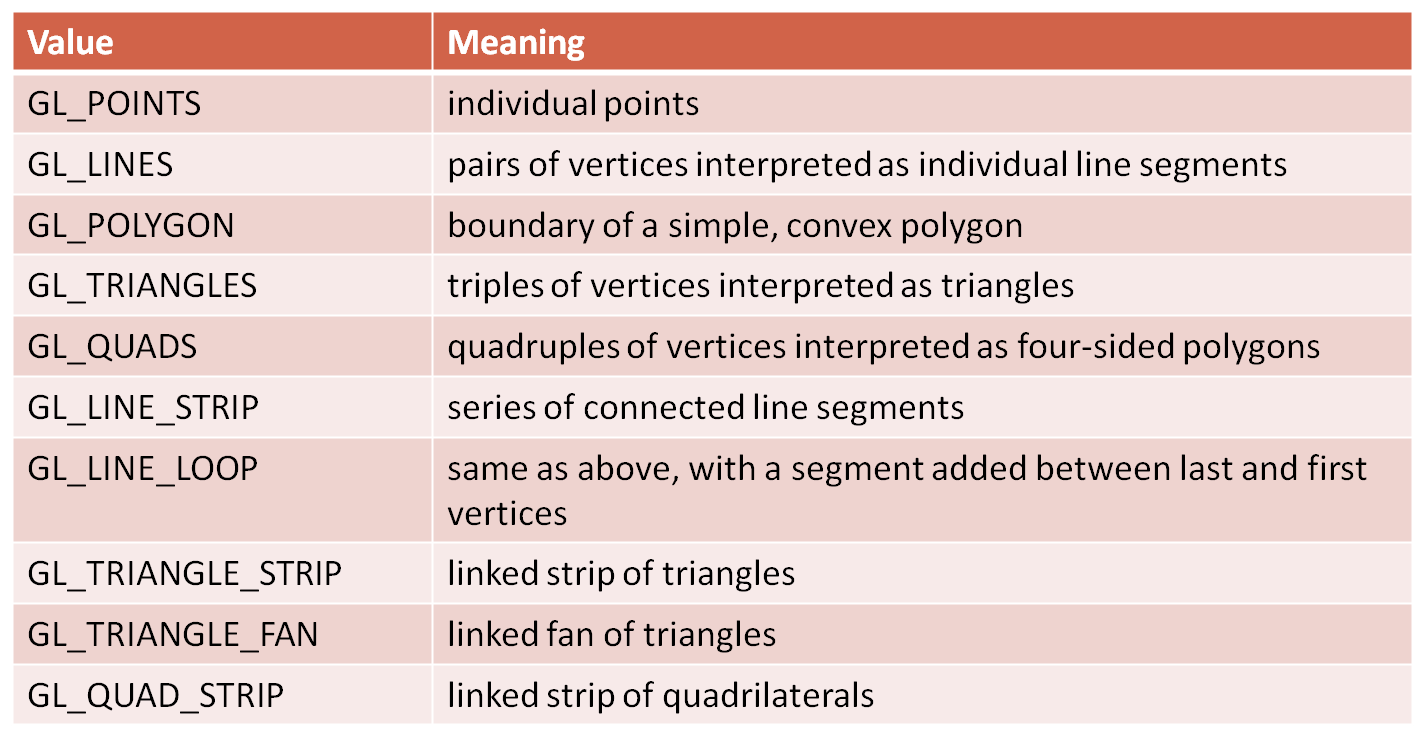
**OpenGL Command Syntax (2)**

****

**OpenGL Command Syntax (3)**

* **glClearColor()** – Specifies the background color
* **glClear()** – Erases the output with background color
* **glMatrixMode()** – Chooses projection/modelview matrix
* **glBegin()/glEnd()** – Model data pumped within this block
* **glVertex()** – Pumps vertex data into OpenGL
* **glViewport()** – Resizes the OpenGL viewport
* **glOrtho()** – Specifies orthogonal view volume
* **glPolygonMode()** – Specifies whether to draw filled polygons or wire-frame polygons

**OpenGL Primitives**

****

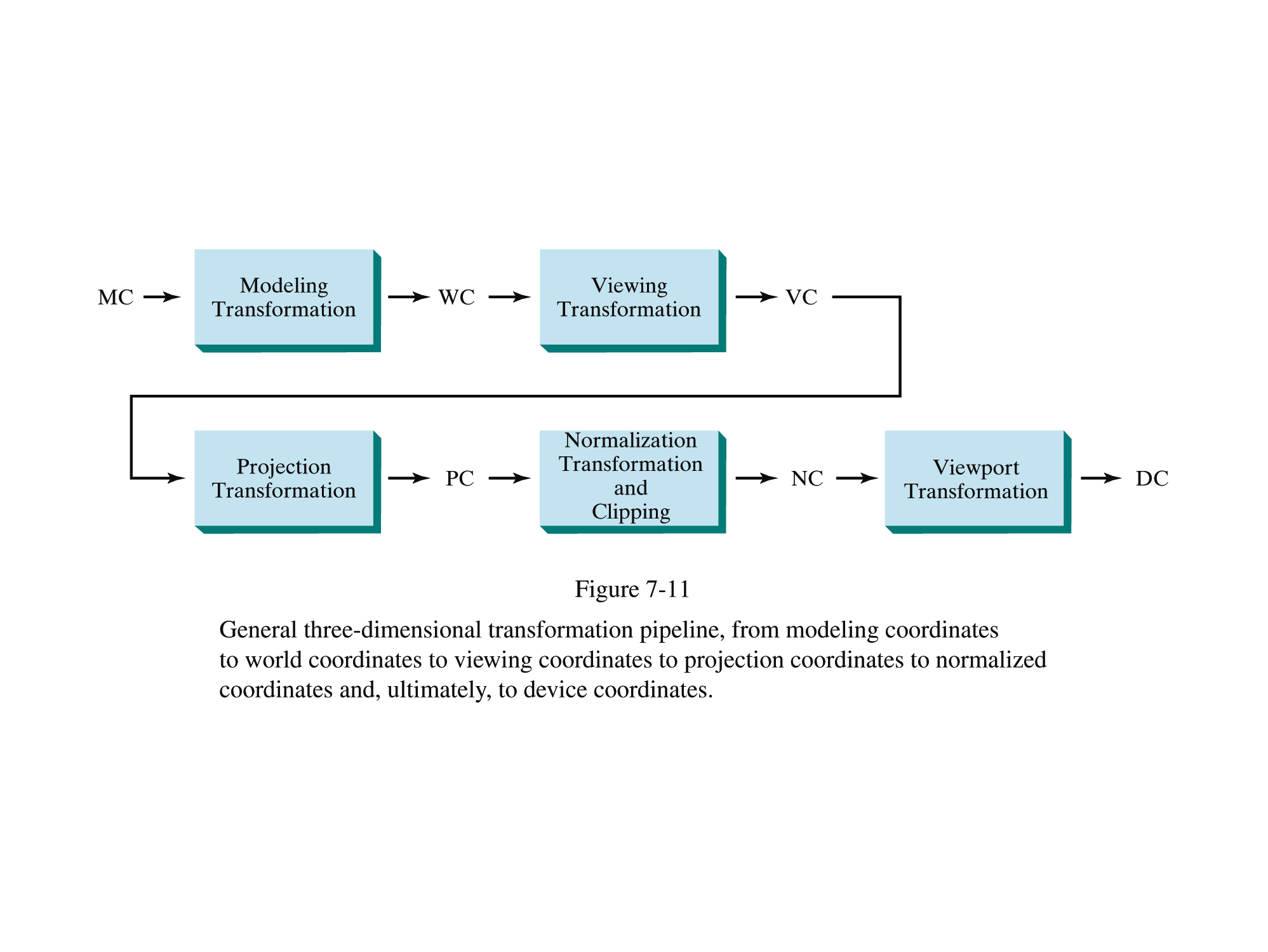
**OpenGL Program Organization**

* **main:** 
  + find GL visual and create window
  + initialize GL states (e.g. viewing, color, lighting)
  + initialize display lists
  + loop
    - check for events (and process them)
    - if window event (window moved, exposed, etc.)
    - modify viewport, if needed
    - redraw
    - else if mouse or keyboard
    - do something, e.g., change states and redraw
* **redraw:** 
  + clear screen (to background color)
  + change state(s), if needed
  + render some graphics
  + change more states
  + render some more graphics

**glMatrixMode**

* glMatrixMode
  + - specify which matrix is the current matrix
* C Specification
  + void glMatrixMode( GLenum *mode* )
* Parameters
  + *mode* Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL\_MODELVIEW, GL\_PROJECTION, and GL\_TEXTURE. The default value is GL\_MODELVIEW.
* Description
  + glMatrixMode sets the current matrix mode. *mode* can assume one of three values: GL\_MODELVIEW Applies subsequent matrix operations to the modelview matrix stack. GL\_PROJECTION Applies subsequent matrix operations to the projection matrix stack.

**General 3D Viewing Pipeline**



* Modeling coordinates (MC)
* World coordinates (WC)
* Viewing coordinates (VC)
* Projection coordinates (PC)
* Normalized coordinates (NC)
* Device coordinates (DC)

**OpenGL 3D Viewing Functions**

* Viewing-transformation function
  + glMatrixMode(GL\_MODELVIEW);
  + gluLookAt(x0,y0,z0,xref,yref,zref,vx,vy,vz);
  + Default: gluLookAt(0,0,0, 0,0,-1, 0,1,0);
  + OpenGL orthogonal-projection function
  + glMatrixMode(GL\_PROJECTION);
  + gluOrtho(xwmin,xwmax, ywmin,ywmax, dnear,dfar);
  + Default: gluOrtho(-1,1, -1,1, -1,1);
  + Note that
    - dnear and dfar must be assigned positive values
    - znear=-dnear and zfar=-dfar
    - The near clipping plane is the view plane

**Open GL program installation and execution steps**

Open GL libraries path

| **File** | **Location** | |
| --- | --- | --- |
| glut32.dll | C:\WINDOWS\System\ |  |
| glut32.lib | C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\lib | |
| glut.h | C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\include\gl | |

* If you plan on giving your program to friends to run using Windows, you must also include the glut32.dll file. If they don't have this file in the same directory as your application or in their C:\WINDOWS\system folder, the program will not run.

**Step 1: Create a Visual Studio 2008 Project**

To create an empty console project in Visual Studio, do the following:

1. Create a new project (File ---> New ---> --->Project)

2. In the Project Types: pane, select Visual C++, Win32. Then select Win 32 Console Application in the Templates: pane. Name your project, select the location for the project and click OK.

3. Click the Application Settings tab on the left, and check the Empty Project box. Then click Finish button

**Step 2: Add Source Code**

1.  Select Project, Add New Item

2. In the Categories pane, select Visual C++, Code. Then select C++ File ( .cpp) in the Templates: pane. Name your file, and then click Add.

**Step 3: Compile and Run the project**

a. Compile the Program

From the Visual Studio's menu Build option (Build  ---> Build Solution)

**b. Execute the program**

From the Visual Studio's menu Debug option (Debug  ---> Start Without Debugging)

**Sample programs**

1. Program to create a simple primitive in Open GL

#include<GL/glut.h>

void mydisplay()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POINTS);

glVertex2f(-0.5,-0.5);

glVertex2f(0.5,0.5);

glVertex2f(-0.5,0.5);

glEnd();

glFlush();

}

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

{

glutInit(&argc, argv);

glutCreateWindow(“Simple”);

glutDisplayFunc(mydisplay);

glutMainLoop();

}

1. Program to create keyboard interface & window sizing

#include<GL/glut.h>

void display()

{ /\* Called when OpenGL needs to update the display\*/

glClearColor(1.0,1.0,1.0,0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

}

void keyboard(unsigned char key, int x, int y)

{ /\* called when a key is pressed \*/

if(key==27) exit(0); /\* 27 is the escape \*/

}

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

{

glutInit(&argc, argv); /\* Initialize OpenGL\*/

glutInitWindowSize(500,500); /\*Set Window size\*/

glutInitWindowPosition(10,10); /\*Set Window Position\*/

glutCreateWindow(“Hai”); /\* Create the window\*/

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glutMainLoop();

}

glutInit(&argc,argv);

glutInitWindowSize(500,500);

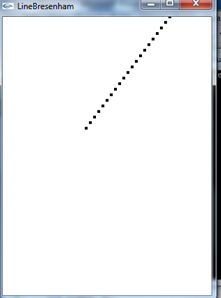
glutInitWindowPosition(100,100);

1. **Write a program to implement Bresenham’s line drawing algorithm with all values of slopes**

**Variant 1:**

#include<stdio.h>  
#include<math.h>  
#include<gl/glut.h>  
GLint X1,Y1,X2,Y2;  
void LineBres(void)  
{  
glClear(GL\_COLOR\_BUFFER\_BIT);  
int dx=abs(X2-X1),dy=abs(Y2-Y1);  
int p=2\*dy-dx;  
int twoDy=2\*dy,twoDyDx=2\*(dy-dx);  
int x,y;  
if(X1>X2)  
{  
x=X2;  
y=Y2;  
X2=X1;  
}  
else  
{  
x=X1;  
y=Y1;  
X2=X2;  
}  
glBegin(GL\_POINTS);  
glVertex2i(x,y);  
while(x<X2)  
{  
x++;  
if(p<0)  
p+=twoDy;  
else  
{  
y++;  
p+=twoDyDx;  
}  
glVertex2i(x,y);  
}  
glEnd();  
glFlush();  
}  
void Init()  
{  
glClearColor(1.0,1.0,1.0,0);  
glColor3f(1.0,0.0,0.0);  
glPointSize(4.0);  
glViewport(0,0,50,50);  
glMatrixMode(GL\_PROJECTION);  
glLoadIdentity();  
gluOrtho2D(0,50,0,50);  
}  
int main(int argc,char \*\*argv)  
{  
printf("enter two points for draw lineBresenham:\n");  
printf("\n enter point1(X1,Y1):");  
scanf("%d,%d",&X1,&Y1); // X1=2 Y1 =2  
printf("\n enter point2(X2,Y2):");  
scanf("%d,%d",&X2,&Y2); // X1=9 X2=11  
glutInit(&argc,argv);  
glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  
glutInitWindowSize(300,400);  
glutInitWindowPosition(0,0);  
glutCreateWindow("Line Bresenham");  
Init();  
glutDisplayFunc(LineBres);  
glutMainLoop();  
return 0;  
}

**Output:**

****

**Variant 2:**

#include <GL/glut.h>

#include <stdio.h>

int x1, y1, x2, y2;

void myInit()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.0, 0.0, 0.0, 1.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 500, 0, 500);

}

void draw\_pixel(int x, int y)

{

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

void draw\_line(int x1, int x2, int y1, int y2)

{

int dx, dy, i, e, x, y, incx, incy, inc1, inc2;

dx = x2-x1;

dy = y2-y1;

if (dx < 0)

dx = -dx;

if (dy < 0)

dy = -dy;

incx = 1;

if (x2 < x1)

incx = -1;

incy = 1;

if (y2 < y1)

incy = -1;

x = x1; y = y1;

if (dx > dy)

{

draw\_pixel(x, y);

e = 2 \* dy-dx;

inc1 = 2\*(dy-dx);

inc2 = 2\*dy;

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

{

if (e >= 0)

{

y += incy;

e += inc1;

}

else

e += inc2;

x += incx;

draw\_pixel(x, y);

}

}

else

{

draw\_pixel(x, y);

e = 2\*dx-dy;

inc1 = 2\*(dx-dy);

inc2 = 2\*dx;

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

{

if (e >= 0)

{

x += incx;

e += inc1;

}

else

e += inc2;

y += incy;

draw\_pixel(x, y);

}

}

}

void myDisplay()

{

draw\_line(x1, x2, y1, y2);

glFlush();

}

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

{

printf( "Enter (x1, y1, x2, y2)\n");

scanf("%d %d %d %d", &x1, &y1, &x2, &y2);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("Bresenham's Line Drawing");

myInit();

glutDisplayFunc(myDisplay);

glutMainLoop();

return 0;

}

**Variant 3:**

#include<GL/glut.h>

#include<stdio.h>

int x1, y1, x2, y2;

void draw\_pixel(int x, int y)

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

void bresenhams\_line\_draw(int x1, int y1, int x2, int y2)

{

float dx = x2 - x1;

float dy = y2 - y1;

float m = dy/dx;

if(m < 1)

{

int decision\_parameter = 2\*dy - dx;

int x = x1; // initial x

int y = y1; // initial y

if(dx < 0) // decide the first point and second point

{

x = x2;

y = y2;

x2 = x1;

}

draw\_pixel(x, y); // plot a point

while(x < x2) // from 1st point to 2nd point

{

if(decision\_parameter >= 0)

{

x = x+1;

y = y+1;

decision\_parameter=decision\_parameter + 2\*dy - 2\*dx \* (y+1 - y);

}

else

{

x = x+1;

y = y;

decision\_parameter = decision\_parameter + 2\*dy - 2\*dx \* (y- y);

}

draw\_pixel(x, y);

}

}

else if(m > 1)

{

int decision\_parameter = 2\*dx - dy;

int x = x1; // initial x

int y = y1; // initial y

if(dy < 0)

{

x = x2;

y = y2;

y2 = y1;

}

draw\_pixel(x, y);

while(y < y2)

{

if(decision\_parameter >= 0)

{

x = x+1;

y = y+1;

decision\_parameter=decision\_parameter + 2\*dx - 2\*dy \* (x+1 - x);

}

else

{

y = y+1;

x = x;

decision\_parameter = decision\_parameter + 2\*dx - 2\*dy \* (x- x);

}

draw\_pixel(x, y);

}

}

else if (m == 1)

{

int x = x1;

int y = y1;

draw\_pixel(x, y);

while(x < x2)

{

x = x+1;

y = y+1;

draw\_pixel(x, y);

}

}

}

void init()

{

glClearColor(1,1,1,1);

gluOrtho2D(0.0, 500.0, 0.0, 500.0); // left ->0, right ->500, bottom ->0, top ->500

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

bresenhams\_line\_draw(x1, y1, x2, y2);

glFlush();

}

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

{

printf( "Enter Start Points (x1,y1)\n");

scanf("%d %d", &x1, &y1); // 1st point from user

printf( "Enter End Points (x2,y2)\n");

scanf("%d %d", &x2, &y2); // 2nd point from user

glutInit(&argc, argv); // initialize graphics system

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB); // single buffered mode with RGB colour variants

glutInitWindowSize(500, 500); // 500 by 500 window size

glutInitWindowPosition(220, 200); // where do you wanna see your window

glutCreateWindow("Bresenham's Line Drawing - FVBIE"); // the title of your window

init(); // initialize the canvas

glutDisplayFunc(display); // call display function

glutMainLoop(); // run forever

}

**Variant 4:**

#include<stdio.h>

#include<graphics.h>

**void** drawline(**int** x0, **int** y0, **int** x1, **int** y1)

{

**int** dx, dy, p, x, y;

dx=x1-x0;

dy=y1-y0;

x=x0;

y=y0;

p=2\*dy-dx;

**while**(x<x1)

{

**if**(p>=0)

{

putpixel(x,y,7);

y=y+1;

p=p+2\*dy-2\*dx;

}

**else**

{

putpixel(x,y,7);

p=p+2\*dy;

}

x=x+1;

}

}

**int** main()

{

**int** gdriver=DETECT, gmode, error, x0, y0, x1, y1;

initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");

**printf**("Enter co-ordinates of first point: ");

scanf("%d%d", &x0, &y0);

**printf**("Enter co-ordinates of second point: ");

scanf("%d%d", &x1, &y1);

drawline(x0, y0, x1, y1);

**return** 0;

}

1. **Create and rotate a triangle about the origin and a fixed point**

**Variant 1:**

#define BLACK 0  
#include <stdio.h>  
#include <math.h>  
#include <GL/glut.h>

GLfloat theta;

GLfloat triangle[3][3]={{100.0,150.0,200.0},{100.0,300.0,100.0},  
{1.0,1.0,1.0}};

GLfloat rotatemat[3][3]={{0},{0},{0}};

GLfloat result[3][3]={{0}, {0}, {0} };

GLfloat arbitrary\_x=100.0;

GLfloat arbitrary\_y=100.0;

GLfloat rotation\_angle;

void multiply()  
{  
int i,j,k;

for(i=0;i<3;i++)  
for(j=0;j<3;j++)  
{  
result[i][j]=0;  
for(k=0;k<3;k++)

result[i][j]=result[i][j]+rotatemat[i][k]\* triangle[k][j];  
}  
}

void rotate()  
{  
GLfloat m,n;  
rotation\_angle=theta\*3.1415/180.0;

m=-  
arbitrary\_x\*(cos(rotation\_angle)-1)+arbitrary\_y\*(sin(rotation\_angle));

n=-arbitrary\_y\*(cos(rotation\_angle)-1)-  
arbitrary\_x\*(sin(rotation\_angle));

rotatemat[0][0]=cos(rotation\_angle);  
rotatemat[0][1]=-sin(rotation\_angle);  
rotatemat[0][2]=m;

rotatemat[1][0]=sin(rotation\_angle);  
rotatemat[1][1]=cos(rotation\_angle);  
rotatemat[1][2]=n;

rotatemat[2][0]=0;  
rotatemat[2][1]=0;  
rotatemat[2][2]=1;  
//multiply the two matrices  
multiply();  
}

void drawtriangle()  
{  
glColor3f(0.0, 0.0, 1.0);

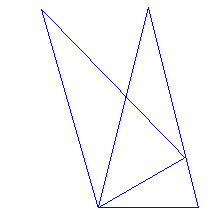
glBegin(GL\_LINE\_LOOP);  
glVertex2f(triangle[0][0], triangle[1][0]);  
glVertex2f(triangle[0][1],triangle[1][1]);  
glVertex2f(triangle[0][2], triangle[1][2]);  
glEnd();  
}

void drawrotatedtriangle()  
{  
glColor3f(0.0, 0.0, 1.0);  
glBegin(GL\_LINE\_LOOP);  
glVertex2f(result[0][0],result[1][0]);  
glVertex2f(result[0][1],result[1][1]);  
glVertex2f(result[0][2],result[1][2]);  
glEnd();  
}  
void display()  
{  
glClearColor(1.0,1.0,1.0,1.0);  
glClear(GL\_COLOR\_BUFFER\_BIT);

drawtriangle();  
rotate();  
drawrotatedtriangle();  
glFlush();  
}  
int main(int argc, char\*\* argv)  
{  
printf("Enter the rotation angle\n");  
scanf("%f", &theta);  
glutInit(&argc,argv);  
glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);  
glutInitWindowSize(600,600);  
glutInitWindowPosition(0,0);  
glutCreateWindow("triangle rotation");  
glutDisplayFunc(display);  
gluOrtho2D(-500.0,499.0,-500.0,499.0);  
glutMainLoop();  
return 0;  
}

**Output:**

**Enter the rotation angle: 30**

****

**Variant 2:**

#include<GL/glut.h>

#include<stdio.h>

int x,y;

int where\_to\_rotate=0;

float rotate\_angle=0;

float translate\_x=0,translate\_y=0;

void draw\_pixel(float x1, float y1)

{

glPointSize(5);

glBegin(GL\_POINTS);

glVertex2f(x1,y1);

glEnd();

}

void triangle(int x, int y)

{

glColor3f(1,0,0);

glBegin(GL\_POLYGON);

glVertex2f(x,y);

glVertex2f(x+400,y+300);

glVertex2f(x+300,y+0);

glEnd();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glColor3f(1,1,1);

draw\_pixel(0,0);

if (where\_to\_rotate == 1)

{

translate\_x = 0;

translate\_y = 0;

rotate\_angle += 1;

}

if (where\_to\_rotate == 2)

{

translate\_x = x;

translate\_y = y;

rotate\_angle += 1;

glColor3f(0,0,1);

draw\_pixel(x,y);

}

glTranslatef(translate\_x, translate\_y, 0);

glRotatef(rotate\_angle, 0, 0, 1);

glTranslatef(-translate\_x, -translate\_y, 0);

triangle(translate\_x,translate\_y);

glutPostRedisplay();

glutSwapBuffers();

}

void init()

{

glClearColor(0,0,0,1);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-800, 800, -800, 800);

glMatrixMode(GL\_MODELVIEW);

}

void rotateMenu (int option)

{

if(option==1)

where\_to\_rotate=1;

if(option==2)

where\_to\_rotate=2;

if(option==3)

where\_to\_rotate=3;

}

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

{

printf( "Enter Fixed Points (x,y) for Rotation: \n");

scanf("%d %d", &x, &y);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB);

glutInitWindowSize(800, 800);

glutInitWindowPosition(0, 0);

glutCreateWindow("Create and Rotate Triangle");

init();

glutDisplayFunc(display);

glutCreateMenu(rotateMenu);

glutAddMenuEntry("Rotate around ORIGIN",1);

glutAddMenuEntry("Rotate around FIXED POINT",2);

glutAddMenuEntry("Stop Rotation",3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

}

**Variant 3:**

#include "stdafx.h"

#include <stdio.h>

#include <math.h>

#include<string.h>

#include <GL/glut.h>

int rot;

void drawtriangle()

{

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINES);

glVertex3f(100,100,1.0);

glVertex3f(100,150,1.0);

glVertex3f(100,150.0,1.0);

glVertex3f(125,125,1.0);

glVertex3f(125,125,1.0);

glVertex3f(100,100,1.0);

glEnd();

}

void rotatefixed()

{

glColor3f(0.0, 0.0, 1.0);

glRasterPos2i(80,80);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'f');

glRasterPos2i(90,80);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'i');

glRasterPos2i(95,80);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'x');

glRasterPos2i(100,80);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'e');

glRasterPos2i(105,80);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'d');

glPushMatrix();

drawtriangle();

glTranslatef(-100.0,-100.0,0.0); // 3. Translate to the origin.

drawtriangle();

glRotatef(-45.0,0.0,0.0,1.0); // 2. Rotate the object.

drawtriangle();

glTranslatef(100.0,100.0,0.0); // 1. Translate to the origin position.

drawtriangle();

glPopMatrix();

}

void rotateorigin()

{

glColor3f(0.0, 0.0, 1.0);

glRasterPos2i(80,80);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'O');

glRasterPos2i(90,80);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'R');

glRasterPos2i(100,80);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'I');

glRasterPos2i(105,80);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'G');

glRasterPos2i(110,80);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'I');

glRasterPos2i(115,80);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'N');

glPushMatrix();

drawtriangle();

glRotatef(-45.0,0.0,0.0,1.0); // 2. Rotate the object.

drawtriangle();

glPopMatrix();

}

void myinit()

{

glClearColor(1.0,1.0,1.0,1.0);

glColor3f(1.0,0.0,0.0);

glPointSize(1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-400.0,400.0,-250.0,250.0);//

}

void xyaxis()

{

glColor3f(1.0, 0.0,0.0);

glRasterPos2i(370,5);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'X');

glRasterPos2i(10,240);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,'Y');

glLineWidth(2.0);

glBegin(GL\_LINES);

glVertex3f(0,250,1.0);

glVertex3f(0,0,1.0);

glEnd();

glColor3f(1.0, 0.0,0.0);

glLineWidth(2.0);

glBegin(GL\_LINES);

glVertex3f(0,0,1.0);

glVertex3f(400,0,1.0);

glEnd();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

//glClearColor3f(0.0,0.0,1.0,0.0);

xyaxis();

if (rot==1)

rotateorigin();

else if (rot==2)

rotatefixed();

glFlush();

}

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

{

if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN) rot=1;

if(btn==GLUT\_RIGHT\_BUTTON&&state==GLUT\_DOWN) rot=2;

display();

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500,500);

glutInitWindowPosition(0,0);

glutCreateWindow("house rotation");

myinit();

glutMouseFunc(mouse);

glutDisplayFunc(display);

glutMainLoop();

}

**Variant 4:**

#include "stdafx.h"

#include <stdio.h>

#include <math.h>

#include<string.h>

#include <GL/glut.h>

int x,y; int rFlag=0;

void draw\_pixel(float x1,float y1)

{

glColor3f(0.0,0.0,1.0);

glPointSize(5.0);

glBegin(GL\_POINTS);

glVertex2f(x1,y1);

glEnd();

}

void triangle()

{ glColor3f(1.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(100,100);

glVertex2f(250,400);

glVertex2f(400,100);

glEnd(); }

float th=0.0;

float trX=0.0,trY=0.0;

void display()

{ glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

if(rFlag==1) //Rotate Around origin

{ trX=0.0; trY=0.0; th+=0.1; draw\_pixel(0.0,0.0); }

if(rFlag==2) //Rotate Around Fixed Point

{ trX=x; trY=y; th+=0.1; draw\_pixel(x,y);

} glTranslatef(trX,trY,0.0);

glRotatef(th,0.0,0.0,1.0);

glTranslatef(-trX,-trY,0.0);

triangle();

glutPostRedisplay();

glutSwapBuffers();

}

void myInit()

{ glClearColor(0.0,0.0,0.0,1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-500.0, 500.0, -500.0, 500.0);

glMatrixMode(GL\_MODELVIEW);

}

void rotateMenu (int option)

{ if(option==1)

rFlag=1; if(option==2)

rFlag=2; if(option==3)

rFlag=3;

}

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

{

printf( "Enter Fixed Points (x,y) for Roration: \n");

scanf("%d %d", &x, &y);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("Create and Rotate Triangle");

myInit();

glutDisplayFunc(display);

glutCreateMenu(rotateMenu);

glutAddMenuEntry("Rotate around ORIGIN",1);

glutAddMenuEntry("Rotate around FIXED POINT",2);

glutAddMenuEntry("Stop Rotation",3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

}

1. **Draw a color cube and spin it using OpenGL transformation matrices**

**Variant 1:**

#include <stdlib.h>  
#include <GL/glut.h>

GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},  
{1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},  
{1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

GLfloat normals[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},  
{1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},  
{1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},  
{1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},  
{1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};

void polygon(int a=0, int b=3, int c=2 , int d=1)  
{

/\* draw a polygon via list of vertices \*/

glBegin(GL\_POLYGON);  
glColor3fv(colors[a]);  
glNormal3fv(normals[a]);  
glVertex3fv(vertices[a]);  
glColor3fv(colors[b]);  
glNormal3fv(normals[b]);  
glVertex3fv(vertices[b]);  
glColor3fv(colors[c]);  
glNormal3fv(normals[c]);  
glVertex3fv(vertices[c]);  
glColor3fv(colors[d]);  
glNormal3fv(normals[d]);  
glVertex3fv(vertices[d]);  
glEnd();  
}

void colorcube(void)  
{

/\* map vertices to faces \*/

polygon(0,3,2,1);  
polygon(2,3,7,6);  
polygon(0,4,7,3);  
polygon(1,2,6,5);  
polygon(4,5,6,7);  
polygon(0,1,5,4);  
}

static GLfloat theta[] = {0.0,0.0,0.0};  
static GLint axis = 2;

void display(void)  
{  
/\* display callback, clear frame buffer and z buffer,  
rotate cube and draw, swap buffers \*/

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
glLoadIdentity();  
glRotatef(theta[0], 1.0, 0.0, 0.0);  
glRotatef(theta[1], 0.0, 1.0, 0.0);  
glRotatef(theta[2], 0.0, 0.0, 1.0);

colorcube();

glFlush();  
glutSwapBuffers();  
}

void spinCube()  
{

/\* Idle callback, spin cube 2 degrees about selected axis \*/

theta[axis] += 1.0;  
if( theta[axis] > 360.0 ) theta[axis] -= 360.0;  
/\* display(); \*/  
glutPostRedisplay();  
}

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

/\* mouse callback, selects an axis about which to rotate \*/

if(btn==GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) axis = 0;  
if(btn==GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN) axis = 1;  
if(btn==GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN) axis = 2;  
}

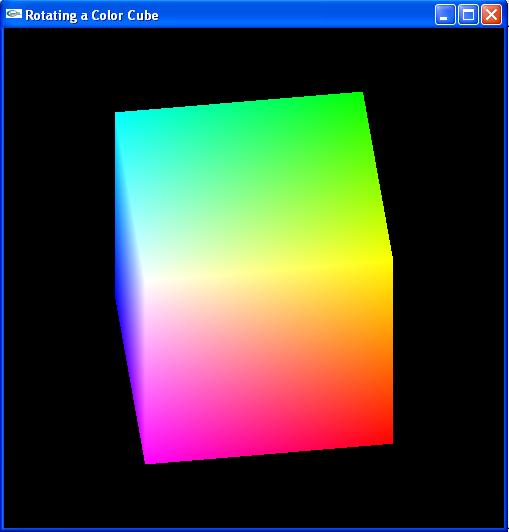
void myReshape(int w, int h)  
{  
glViewport(0, 0, w, h);  
glMatrixMode(GL\_PROJECTION);  
glLoadIdentity();  
if (w <= h)  
glOrtho(-2.0, 2.0, -2.0 \* (GLfloat) h / (GLfloat) w,  
2.0 \* (GLfloat) h / (GLfloat) w, -10.0, 10.0);  
else  
glOrtho(-2.0 \* (GLfloat) w / (GLfloat) h,  
2.0 \* (GLfloat) w / (GLfloat) h, -2.0, 2.0, -10.0, 10.0);  
glMatrixMode(GL\_MODELVIEW);  
}

int main(int argc, char \*\*argv)  
{  
// glutInit(&argc, argv);

/\* need both double buffering and z buffer \*/

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);  
glutInitWindowSize(500, 500);  
glutCreateWindow("Rotating a Color Cube");  
glutReshapeFunc(myReshape);  
glutDisplayFunc(display);  
glutIdleFunc(spinCube);  
glutMouseFunc(mouse);  
glEnable(GL\_DEPTH\_TEST); /\* Enable hidden--surface--removal \*/  
glutMainLoop();  
return 0;  
}

**Output:**

****

**Variant 2:**

#include <GL/glut.h>

GLfloat vertices[][3]=

{

{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},

{1.0,1.0,-1.0},{-1.0,1.0,-1.0},

{-1.0,-1.0,1.0},{1.0,-1.0,1.0},

{1.0,1.0,1.0},{-1.0,1.0,1.0}

};

GLfloat colors[][3]=

{

{0.0,0.0,0.0},{0,0.0,0.0},

{1.0,1.0,0.0},{0.0,1.0,1.0},

{0.0,0.0,1.0},{1.0,0.0,1.0},

{1.0,1.0,1.0},{0.0,1.0,1.0}

};

void polygon(int a,int b,int c,int d)

{

glBegin(GL\_POLYGON);

glColor3fv(colors[a]);

glVertex3fv(vertices[a]);

glColor3fv(colors[b]);

glVertex3fv(vertices[b]);

glColor3fv(colors[c]);

glVertex3fv(vertices[c]);

glColor3fv(colors[d]);

glVertex3fv(vertices[d]);

glEnd();

}

void colorcube(void)

{

polygon(0,3,2,1);

polygon(2,3,7,6);

polygon(0,4,7,3);

polygon(1,2,6,5);

polygon(4,5,6,7);

polygon(0,1,5,4);

//similarly for remaining 5 faces

}

static GLfloat theta[]={0.0,0.0,0.0};//angle of rotation along each axis

static GLint axis=2;//current axis

void display(void)

{

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

glLoadIdentity();

glRotatef(theta[0],1.0,0.0,0.0);//for x axis

glRotatef(theta[1],0.0,1.0,0.0);

glRotatef(theta[2],0.0,0.0,1.0);

// similarly for y and z axis

colorcube();

glutSwapBuffers();

}

void spinCube()

{

theta[axis]+=5.0;

if(theta[axis]>360.0) theta[axis]-=360.0;

glutPostRedisplay();

}

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

{

if(btn==GLUT\_LEFT\_BUTTON&&state==GLUT\_DOWN)axis=0;//x axis rotation

if(btn==GLUT\_MIDDLE\_BUTTON&&state==GLUT\_DOWN)axis=1;//y axis rotn

if(btn==GLUT\_RIGHT\_BUTTON&&state==GLUT\_DOWN)axis=2;

//similarly for y and z axis

}

void myReshape(int w,int h)// common reshape func for all prgrms

{

glViewport(0,0,w,h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if(w<=h)

glOrtho(-2.0,2.0,-2.0\*(GLfloat)h/(GLfloat)w,2.0\*(GLfloat)h/(GLfloat)w,-

10.0,10.0);

else

glOrtho(-2.0\*(GLfloat)w/(GLfloat)h,2.0\*(GLfloat)w/(GLfloat)h,-2.0,2.0,-

10.0,10.0);

glMatrixMode(GL\_MODELVIEW);

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(500,500);

glutCreateWindow("color cube viewer");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutIdleFunc(spinCube);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

return 0;

}

**Variant 3:**

#include <stdlib.h>

#include <GL/glut.h>

GLfloat vertices[] = {-1.0,-1.0,-1.0,1.0,-1.0,-1.0,

1.0,1.0,-1.0, -1.0,1.0,-1.0, -1.0,-1.0,1.0,

1.0,-1.0,1.0, 1.0,1.0,1.0, -1.0,1.0,1.0};

GLfloat colors[] = {0.0,0.0,0.0,1.0,0.0,0.0,

1.0,1.0,0.0, 0.0,1.0,0.0, 0.0,0.0,1.0,

1.0,0.0,1.0, 1.0,1.0,1.0, 0.0,1.0,1.0};

GLubyte cubeIndices[]={0,3,2,1,2,3,7,6,0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4};

static GLfloat theta[] = {0.0,0.0,0.0};

static GLint axis = 2;

void display(void)

{

/\* display callback, clear frame buffer and z buffer,

rotate cube and draw, swap buffers \*/

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

glLoadIdentity();

glRotatef(theta[0], 1.0, 0.0, 0.0);

glRotatef(theta[1], 0.0, 1.0, 0.0);

glRotatef(theta[2], 0.0, 0.0, 1.0);

glDrawElements(GL\_QUADS, 24, GL\_UNSIGNED\_BYTE, cubeIndices);

/\*glBegin(GL\_LINES);

glVertex3f(0.0,0.0,0.0);

glVertex3f(1.0,1.0,1.0);

glEnd(); \*/

glFlush();

glutSwapBuffers();

}

void spinCube()

{

/\* Idle callback, spin cube 2 degrees about selected axis \*/

theta[axis] += 2.0;

if( theta[axis] > 360.0 ) theta[axis] -= 360.0;

glutPostRedisplay();

}

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

{

/\* mouse callback, selects an axis about which to rotate \*/

if(btn==GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) axis = 0;

if(btn==GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN) axis = 1;

if(btn==GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN) axis = 2;

}

void myReshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-2.0, 2.0, -2.0, 2.0, -10.0, 10.0);

glMatrixMode(GL\_MODELVIEW);

}

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

{

/\* need both double buffering and z buffer \*/

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutCreateWindow("Spin a colorcube");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

glutIdleFunc(spinCube);

glutMouseFunc(mouse);

glEnable(GL\_DEPTH\_TEST); /\* Enable hidden--surface--removal \*/

glEnableClientState(GL\_COLOR\_ARRAY);

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(3, GL\_FLOAT, 0, vertices);

glColorPointer(3,GL\_FLOAT, 0, colors);

glColor3f(1.0,1.0,1.0);

glutMainLoop();

}

**Variant 4:**

#include<stdlib.h>

#include<GL/glut.h>

GLfloat vertices[] = { -1, -1, -1,

1, -1, -1,

1, 1, -1,

-1, 1, -1,

-1, -1, 1,

1, -1, 1,

1, 1, 1,

-1, 1, 1

};

GLfloat colors[] = { 0, 0, 0, // white color

1, 0, 0, // red color .. so on for eight faces of cube

1, 1, 0,

0, 1, 0,

0, 0, 1,

1, 0, 1,

1, 1, 1,

0, 1, 1

};

GLubyte cubeIndices[] = {0, 3, 2, 1,

2, 3, 7, 6,

0, 4, 7, 3,

1, 2, 6, 5,

4, 5, 6, 7,

0, 1, 5, 4

};

static GLfloat theta[]= {0, 0, 0}; // initial angles

static GLint axis=2; // let us assume the right mouse button has been clicked initially

void display(void)

{

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

glLoadIdentity();

glRotatef (theta[0], 1, 0, 0); // first angle rotation via x axis

glRotatef (theta[1], 0, 1, 0); // second angle rotation via y axis

glRotatef (theta[2], 0, 0, 1); // third angle rotation via z axis

glDrawElements(GL\_QUADS,24,GL\_UNSIGNED\_BYTE,cubeIndices); // draw the cube

glutSwapBuffers(); // show the output

}

void spinCube()

{

theta[axis] += 2; // rotate every 2 degrees

if (theta[axis] > 360) // it the rotation angle crosses 360 degrees, make it 0 degree

theta[axis] -= 360;

glutPostRedisplay(); // call display again

}

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

{

if (btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN)

axis=0; // x axis rotation

if (btn==GLUT\_MIDDLE\_BUTTON && state==GLUT\_DOWN)

axis=1; // y axis rotation

if (btn==GLUT\_RIGHT\_BUTTON && state==GLUT\_DOWN)

axis=2; // z axis rotation

}

void myReshape(int w, int h)

{

glViewport(0,0,w,h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if(w<=h)

glOrtho (-2, 2, -2\*(GLfloat)h/(GLfloat)w, 2\*(GLfloat)h / (GLfloat)w, -10, 10);

else

glOrtho (-2\*(GLfloat)w/(GLfloat)h, 2\*(GLfloat)w / (GLfloat)h, -2, 2, -10, 10);

glMatrixMode(GL\_MODELVIEW);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutCreateWindow("Spin a color cube");

glutReshapeFunc(myReshape); // calls myReshape whenever we change the window size

glutDisplayFunc(display); // call display function

glutIdleFunc(spinCube); // whenever we are idle, calls spinCube function

glutMouseFunc(mouse); // calls mouse function whenever we interact with mouse

glEnable(GL\_DEPTH\_TEST); // enables depth – for 3D

glEnableClientState(GL\_COLOR\_ARRAY); // enables colour and vertex properties

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(3, GL\_FLOAT, 0, vertices); // glVertexPointer(size,type,stride,pointer)

glColorPointer(3, GL\_FLOAT, 0, colors); // glColorPointer(size,type,stride,pointer)

glColor3f(1, 1, 1);

glutMainLoop();

}

1. **Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing**

Variant 1:

#include<stdlib.h>  
#include<GL/glut.h>  
GLfloat vertices[][3]={{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},  
{1.0,1.0,-1.0},{-1.0,1.0,-1.0},{-1.0,-1.0,1.0},  
{1.0,-1.0,1.0},{1.0,1.0,1.0},{-1.0,1.0,1.0}};

GLfloat colors[][3]={{0.0,0.0,0.0},{1.0,0.0,0.0},  
{1.0,1.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},  
{1.0,0.0,1.0},{1.0,1.0,1.0},{0.0,1.0,1.0}};

void polygon(int a, int b, int c, int d)  
{  
glBegin(GL\_POLYGON);  
glColor3fv(colors[a]);  
glVertex3fv(vertices[a]);  
glColor3fv(colors[b]);  
glVertex3fv(vertices[b]);  
glColor3fv(colors[c]);  
glVertex3fv(vertices[c]);  
glColor3fv(colors[d]);  
glVertex3fv(vertices[d]);  
glEnd();  
}

void colorcube()  
{  
polygon(0,3,2,1);  
polygon(2,3,7,6);  
polygon(0,4,7,3);  
polygon(1,2,6,5);  
polygon(4,5,6,7);  
polygon(0,1,5,4);  
}

static GLfloat theta[]={0.0,0.0,0.0};  
static GLint axis=2;  
static GLdouble viewer[]={0.0,0.0,5.0};

void display(void)  
{  
glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);  
glLoadIdentity();  
gluLookAt(viewer[0], viewer[1],viewer[2],0.0,0.0,0.0,0.0,1.0,0.0);  
glRotatef(theta[0],1.0,0.0,0.0);  
glRotatef(theta[1],0.0,1.0,0.0);  
glRotatef(theta[2],0.0,0.0,1.0);  
colorcube();  
glFlush();  
glutSwapBuffers();  
}

void mouse(int btn, int state, int x, int y)  
{  
if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN) axis=0;  
if(btn==GLUT\_MIDDLE\_BUTTON&&state==GLUT\_DOWN) axis=1;  
if(btn==GLUT\_RIGHT\_BUTTON&&state==GLUT\_DOWN) axis=2;  
theta[axis]+=2.0;  
if(theta[axis]>360.0) theta[axis]-=360.0;  
display();  
}

void keys(unsigned char key, int x, int y)  
{  
if(key == 'x') viewer[0]-=1.0;  
if(key == 'X') viewer[0]+=1.0;  
if(key == 'y') viewer[1]-=1.0;  
if(key == 'Y') viewer[1]+=1.0;  
if(key == 'z') viewer[2]-=1.0;  
if(key == 'Z') viewer[2]+=1.0;  
display();  
}

void myReshape(int w, int h)  
{  
glViewport(0,0,w,h);  
glMatrixMode(GL\_PROJECTION);  
glLoadIdentity();  
if(w<=h)  
glFrustum(-2.0,2.0,-2.0\*(GLfloat)h/(GLfloat)w,2.0\*(GLfloat)h/(GLfloat)w, 2.0,20.0);  
else  
glFrustum(-2.0,2.0,-2.0\*(GLfloat)w/(GLfloat)h,2.0\*(GLfloat)w/(GLfloat)h, 2.0,20.0);  
glMatrixMode(GL\_MODELVIEW);  
}

int main(int argc, char \*\*argv)  
{  
glutInit(&argc, argv);  
glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);  
glutInitWindowSize(500,500);  
glutCreateWindow("Colorcube Viewer");  
glutReshapeFunc(myReshape);  
glutDisplayFunc(display);  
glutMouseFunc(mouse);  
glutKeyboardFunc(keys);  
glEnable(GL\_DEPTH\_TEST);  
glutMainLoop();  
return 0;  
}

**Output:**

****

**Variant 2:**

#include<stdlib.h>

#include<GL/glut.h>

GLfloat vertices[][3]={{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},

{1.0,1.0,-1.0},{-1.0,1.0,-1.0},{-1.0,-1.0,1.0},

{1.0,-1.0,1.0},{1.0,1.0,1.0},{-1.0,1.0,1.0}};

GLfloat colors[][3]={{0.0,0.0,0.0},{1.0,0.0,0.0},

{1.0,1.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},

{1.0,0.0,1.0},{1.0,1.0,1.0},{0.0,1.0,1.0}};

**void colorcube()**

**{**

polygon(0,3,2,1);

polygon(2,3,7,6);

polygon(0,4,7,3);

polygon(1,2,6,5);

polygon(4,5,6,7);

polygon(0,1,5,4);

**}**

**void polygon(int a, int b, int c, int d)**

**{**

glBegin(GL\_POLYGON);

glColor3fv(colors[a]);

glVertex3fv(vertices[a]);

glColor3fv(colors[b]);

glVertex3fv(vertices[b]);

glColor3fv(colors[c]);

glVertex3fv(vertices[c]);

glColor3fv(colors[d]);

glVertex3fv(vertices[d]);

glEnd();

**}**

**void colorcube()**

**{**

polygon(0,3,2,1);

polygon(2,3,7,6);

polygon(0,4,7,3);

polygon(1,2,6,5);

polygon(4,5,6,7);

polygon(0,1,5,4);

**}**

static GLfloat theta[]={0.0,0.0,0.0};

static GLint axis=2;

static GLdouble viewer[]={0.0,0.0,5.0};

**void display(void)**

**{**

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

glLoadIdentity();

gluLookAt(viewer[0], viewer[1],viewer[2],0.0,0.0,0.0,

0.0,1.0,0.0);

glRotatef(theta[0],1.0,0.0,0.0);

glRotatef(theta[1],0.0,1.0,0.0);

glRotatef(theta[2],0.0,0.0,1.0);

colorcube();

glFlush();

glutSwapBuffers();

**}**

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

**{**

if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN) axis=0;

if(btn==GLUT\_MIDDLE\_BUTTON&&state==GLUT\_DOWN) axis=1;

if(btn==GLUT\_RIGHT\_BUTTON&&state==GLUT\_DOWN) axis=2;

theta[axis]+=2.0;

if(theta[axis]>360.0) theta[axis]-=360.0;

display();

**}**

**void keys(unsigned char key, int x, int y)**

**{**

if(key == 'x') viewer[0]-=1.0;

if(key == 'X') viewer[0]+=1.0;

if(key == 'y') viewer[1]-=1.0;

if(key == 'Y') viewer[1]+=1.0;

if(key == 'z') viewer[2]-=1.0;

if(key == 'Z') viewer[2]+=1.0;

display();

**}**

**void myReshape(int w, int h)**

**{**

glViewport(0,0,w,h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if(w<=h)

glFrustum(-2.0,2.0,-2.0\*(GLfloat)h/(GLfloat)w,

2.0\*(GLfloat)h/(GLfloat)w, 2.0,20.0);

else

glFrustum(-2.0,2.0,-2.0\*(GLfloat)w/(GLfloat)h,

2.0\*(GLfloat)w/(GLfloat)h, 2.0,20.0);

glMatrixMode(GL\_MODELVIEW);

**}**

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

**{**

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(500,500);

glutCreateWindow("Colorcube Viewer");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutKeyboardFunc(keys);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

**}**

**Variant 3:**

#include

#include

GLfloat vertices[][3]={{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},

{1.0,1.0,-1.0},{-1.0,1.0,-1.0},{-1.0,-1.0,1.0},

{1.0,-1.0,1.0},{1.0,1.0,1.0},{-1.0,1.0,1.0}};

Glfloat colors[][3]={{0.0,0.0,0.0},{1.0,0.0,0.0},

{1.0,1.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},

{1.0,0.0,1.0},{1.0,1.0,1.0},{0.0,1.0,1.0}};

void polygon(int a, int b, int c, int d)

{

glBegin(GL\_POLYGON);

glColor3fv(colors[a]);

glVertex3fv(vertices[a]);

glColor3fv(colors[b]);

glVertex3fv(vertices[b]);

glColor3fv(colors[c]);

glVertex3fv(vertices[c]);

glColor3fv(colors[d]);

glVertex3fv(vertices[d]);

glEnd();

}

void colorcube()

{

polygon(0,3,2,1);

polygon(2,3,7,6);

polygon(0,4,7,3);

polygon(1,2,6,5);

polygon(4,5,6,7);

polygon(0,1,5,4);

}

static GLfloat theta[]={0.0,0.0,0.0};

static GLint axis=2;

static GLdouble viewer[]={0.0,0.0,5.0};

void display(void)

{

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

glLoadIdentity();

gluLookAt(viewer[0], viewer[1],viewer[2],0.0,0.0,0.0,

0.0,1.0,0.0);

glRotatef(theta[0],1.0,0.0,0.0);

glRotatef(theta[1],0.0,1.0,0.0);

glRotatef(theta[2],0.0,0.0,1.0);

colorcube();

glFlush();

glutSwapBuffers();

}

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

{

if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN) axis=0;

if(btn==GLUT\_MIDDLE\_BUTTON&&state==GLUT\_DOWN) axis=1;

if(btn==GLUT\_RIGHT\_BUTTON&&state==GLUT\_DOWN) axis=2;

theta[axis]+=2.0;

if(theta[axis]>360.0) theta[axis]-=360.0;

display();

}

void keys(unsigned char key, int x, int y)

{

if(key == 'x') viewer[0]-=1.0;

if(key == 'X') viewer[0]+=1.0;

if(key == 'y') viewer[1]-=1.0;

if(key == 'Y') viewer[1]+=1.0;

if(key == 'z') viewer[2]-=1.0;

if(key == 'Z') viewer[2]+=1.0;

display();

}

void myReshape(int w, int h)

{

glViewport(0,0,w,h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if(w<=h)

glFrustum(-2.0,2.0,-2.0\*(GLfloat)h/(GLfloat)w, 2.0\*(GLfloat)h/(GLfloat)w, 2.0,20.0);

else

glFrustum(-2.0,2.0,-2.0\*(GLfloat)w/(GLfloat)h, 2.0\*(GLfloat)w/(GLfloat)h, 2.0,20.0);

glMatrixMode(GL\_MODELVIEW);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(500,500);

glutCreateWindow("Colorcube Viewer");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutKeyboardFunc(keys);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

**Variant 4:**

|  |
| --- |
| #include "stdafx.h" |
|  | #include<GL/glut.h> |
|  | float ver[8][3]={{0,0,0},{1,0,0},{1,1,0},{0,1,0},{0,0,1},{1,0,1},{1,1,1},{0,1,1}}; |
|  | float v1[3]={0,0,5}; |
|  | void polygon(int a,int b,int c,int d); |
|  | void polygon1(); |
|  | void display() |
|  | { |
|  | glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT); |
|  | glLoadIdentity(); |
|  | gluLookAt(v1[0],v1[1],v1[2],0,0,0,0,1,0); |
|  | polygon1(); |
|  | glFlush(); |
|  | } |
|  | void init() |
|  | { |
|  | glClearColor(0.0,0.0,0.0,1.0); |
|  | } |
|  | void polygon1() |
|  | { |
|  | polygon(0,1,2,3); |
|  | polygon(4,5,6,7); |
|  | polygon(5,1,2,6); |
|  | polygon(4,0,3,7); |
|  | polygon(4,5,1,0); |
|  | polygon(7,6,2,3); |
|  | } |
|  | void polygon(int a,int b,int c,int d) |
|  | { |
|  | glBegin(GL\_POLYGON); |
|  | glColor3fv(ver[a]); |
|  | glVertex3fv(ver[a]); |
|  | glColor3fv(ver[b]); |
|  | glVertex3fv(ver[b]); |
|  | glColor3fv(ver[c]); |
|  | glVertex3fv(ver[c]); |
|  | glColor3fv(ver[d]); |
|  | glVertex3fv(ver[d]); |
|  | glEnd(); |
|  | } |
|  | void key(unsigned char f,int x,int y) |
|  | { |
|  | if(f=='x')v1[0]-=.10; |
|  | if(f=='X')v1[0]+=.10; |
|  | if(f=='y')v1[1]-=.10; |
|  | if(f=='Y')v1[1]+=.10; |
|  | if(f=='z')v1[2]-=.10; |
|  | if(f=='Z')v1[2]+=.10; |
|  | display(); |
|  | } |
|  | void Reshape(int w,int h) |
|  | { |
|  | glViewport(0,0,w,h); |
|  | glMatrixMode(GL\_PROJECTION); |
|  | glLoadIdentity(); |
|  | if(w<=h) |
|  | glFrustum(-2.0,2.0,-2.0\*w/h,2.0\*w/h,2.0,20); |
|  | else |
|  | glFrustum(-2.0,2.0,-2.0\*w/h,2.0\*w/h,2.0,20); |
|  | glMatrixMode(GL\_MODELVIEW); |
|  | } |
|  | void main() |
|  | { |
|  | glutInitDisplayMode(GLUT\_SINGLE|GLUT\_DEPTH); |
|  | glutInitWindowSize(500,500); |
|  | glutInitWindowPosition(10,10); |
|  | glutCreateWindow("AIET"); |
|  | init(); |
|  | glutDisplayFunc(display); |
|  | glutKeyboardFunc(key); |
|  | glEnable(GL\_DEPTH\_TEST); |
|  | glutReshapeFunc(Reshape); |
|  | glutMainLoop(); |
|  | } |

1. **Clip a lines using Cohen-Sutherland algorithm**

**Variant 1:**

#include <stdio.h>  
#include <GL/glut.h>  
#define outcode int

double xmin=50,ymin=50, xmax=100,ymax=100; // Window boundaries  
double xvmin=200,yvmin=200,xvmax=300,yvmax=300; // Viewport boundaries

//bit codes for the right, left, top, & bottom  
const int RIGHT = 2;  
const int LEFT = 1;  
const int TOP = 8;  
const int BOTTOM = 4;

//used to compute bit codes of a point  
outcode ComputeOutCode (double x, double y);

//Cohen-Sutherland clipping algorithm clips a line from  
//P0 = (x0, y0) to P1 = (x1, y1) against a rectangle with  
//diagonal from (xmin, ymin) to (xmax, ymax).

void CohenSutherlandLineClipAndDraw (double x0, double y0,double x1, double y1)  
{  
//Outcodes for P0, P1, and whatever point lies outside the clip rectangle  
outcode outcode0, outcode1, outcodeOut;  
bool accept = false, done = false;

//compute outcodes  
outcode0 = ComputeOutCode (x0, y0);  
outcode1 = ComputeOutCode (x1, y1);  
do{  
if (!(outcode0 | outcode1)) //logical or is 0 Trivially accept & exit  
{  
accept = true;  
done = true;  
}  
else if (outcode0 & outcode1) //logical and is not 0. Trivially reject and exit  
done = true;  
else  
{  
//failed both tests, so calculate the line segment to clip  
//from an outside point to an intersection with clip edge  
double x, y;

//At least one endpoint is outside the clip rectangle; pick it.  
outcodeOut = outcode0? outcode0: outcode1;

//Now find the intersection point;  
//use formulas y = y0 + slope \* (x - x0), x = x0 + (1/slope)\* (y - y0)  
if (outcodeOut & TOP) //point is above the clip rectangle  
{  
x = x0 + (x1 - x0) \* (ymax - y0)/(y1 - y0);  
y = ymax;  
}  
else if (outcodeOut & BOTTOM) //point is below the clip rectangle  
{  
x = x0 + (x1 - x0) \* (ymin - y0)/(y1 - y0);  
y = ymin;  
}  
else if (outcodeOut & RIGHT) //point is to the right of clip rectangle  
{  
y = y0 + (y1 - y0) \* (xmax - x0)/(x1 - x0);  
x = xmax;  
}  
else //point is to the left of clip rectangle  
{  
y = y0 + (y1 - y0) \* (xmin - x0)/(x1 - x0);  
x = xmin;  
}

//Now we move outside point to intersection point to clip  
//and get ready for next pass.  
if (outcodeOut == outcode0)  
{  
x0 = x;  
y0 = y;  
outcode0 = ComputeOutCode (x0, y0);  
}  
else  
{  
x1 = x;  
y1 = y;  
outcode1 = ComputeOutCode (x1, y1);  
}  
}  
}while (!done);

if (accept)  
{ // Window to viewport mappings  
double sx=(xvmax-xvmin)/(xmax-xmin); // Scale parameters  
double sy=(yvmax-yvmin)/(ymax-ymin);  
double vx0=xvmin+(x0-xmin)\*sx;  
double vy0=yvmin+(y0-ymin)\*sy;  
double vx1=xvmin+(x1-xmin)\*sx;  
double vy1=yvmin+(y1-ymin)\*sy;  
//draw a red colored viewport  
glColor3f(1.0, 0.0, 0.0);  
glBegin(GL\_LINE\_LOOP);  
glVertex2f(xvmin, yvmin);  
glVertex2f(xvmax, yvmin);  
glVertex2f(xvmax, yvmax);  
glVertex2f(xvmin, yvmax);  
glEnd();

glColor3f(0.0,0.0,1.0); // draw blue colored clipped line  
glBegin(GL\_LINES);  
glVertex2d (vx0, vy0);  
glVertex2d (vx1, vy1);  
glEnd();  
}  
}  
//Compute the bit code for a point (x, y) using the clip rectangle  
//bounded diagonally by (xmin, ymin), and (xmax, ymax)  
outcode ComputeOutCode (double x, double y)  
{  
outcode code = 0;  
if (y > ymax) //above the clip window  
code |= TOP;  
else if (y < ymin) //below the clip window  
code |= BOTTOM;  
if (x > xmax) //to the right of clip window  
code |= RIGHT;  
else if (x < xmin) //to the left of clip window  
code |= LEFT;  
return code;  
}

void display()  
{  
double x0=60,y0=20,x1=80,y1=120;  
glClear(GL\_COLOR\_BUFFER\_BIT);  
//draw the line with red color  
glColor3f(1.0,0.0,0.0);  
//bres(120,20,340,250);  
glBegin(GL\_LINES);  
glVertex2d (x0, y0);  
glVertex2d (x1, y1);  
glEnd();

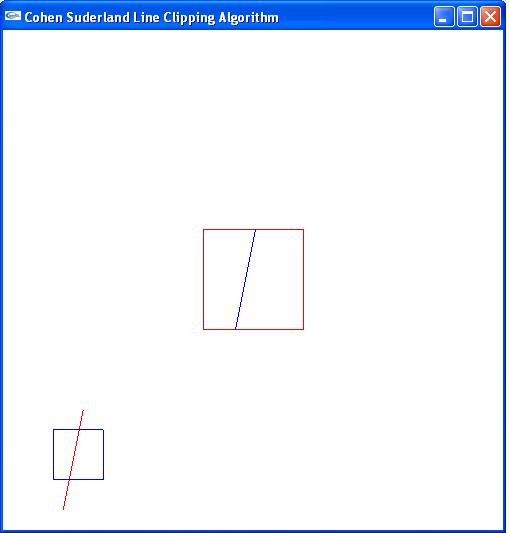
//draw a blue colored window  
glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_LOOP);  
glVertex2f(xmin, ymin);  
glVertex2f(xmax, ymin);  
glVertex2f(xmax, ymax);  
glVertex2f(xmin, ymax);  
glEnd();

CohenSutherlandLineClipAndDraw(x0,y0,x1,y1);  
glFlush();  
}

void myinit()  
{  
glClearColor(1.0,1.0,1.0,1.0);  
glColor3f(1.0,0.0,0.0);  
glPointSize(1.0);  
glMatrixMode(GL\_PROJECTION);  
glLoadIdentity();  
gluOrtho2D(0.0,499.0,0.0,499.0);  
}  
int main(int argc, char\*\* argv)  
{  
glutInit(&argc,argv);  
glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);  
glutInitWindowSize(500,500);  
glutInitWindowPosition(0,0);  
glutCreateWindow("Cohen Suderland Line Clipping Algorithm");  
glutDisplayFunc(display);  
myinit();  
glutMainLoop();  
return 0;  
}

**Output:**

****

**Variant 2:**

#include <GL/glut.h>

#define SCREEN\_WIDTH 640

#define SCREEN\_HEIGHT 480

typedef struct {

GLfloat x, y;

} Point;

const GLint WIN\_LEFT\_BIT = 0x01;

const GLint WIN\_RIGHT\_BIT = 0x02;

const GLint WIN\_BOTTOM\_BIT = 0x04;

const GLint WIN\_TOP\_BIT = 0x08;

void init\_graph(int argc, char \*\*argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(SCREEN\_WIDTH, SCREEN\_HEIGHT);

glutCreateWindow(argv[0]);

glClearColor(1.0, 1.0, 1.0, 0.0);

glPointSize(1.0f);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, SCREEN\_WIDTH, 0, SCREEN\_HEIGHT);

}

void close\_graph() {

glutMainLoop();

}

void swap\_points(Point \*p1, Point \*p2) {

Point t = \*p1;

\*p1 = \*p2;

\*p2 = t;

}

void swap\_codes(GLint \*x, GLint \*y) {

GLint t = \*x;

\*x = \*y;

\*y = t;

}

GLint inside(GLint code) {

return !code;

}

GLint accept(GLint code1, GLint code2) {

return !(code1 | code2);

}

GLint reject(GLint code1, GLint code2) {

return code1 & code2;

}

GLint encode(Point p1, Point win\_min, Point win\_max) {

GLint code = 0x00;

if (p1.x < win\_min.x) code |= WIN\_LEFT\_BIT;

if (p1.x > win\_max.x) code |= WIN\_RIGHT\_BIT;

if (p1.y < win\_min.y) code |= WIN\_BOTTOM\_BIT;

if (p1.y > win\_max.y) code |= WIN\_TOP\_BIT;

return code;

}

GLint round(GLfloat a) {

return (GLint) (a + 0.5f);

}

void line\_clip(Point p1, Point p2, Point win\_min, Point win\_max) {

GLint code1, code2;

GLint done = 0, plot\_line = 0;

GLfloat m = 0;

if (p1.x != p2.x) {

m = (p2.y - p1.y) / (p2.x - p1.x);

}

while (!done) {

code1 = encode(p1, win\_min, win\_max);

code2 = encode(p2, win\_min, win\_max);

if (accept(code1, code2)) {

done = 1;

plot\_line = 1;

} else if (reject(code1, code2)) {

done = 1;

} else {

if (inside(code1)) {

swap\_points(&p1, &p2);

swap\_codes(&code1, &code2);

}

if (code1 & WIN\_LEFT\_BIT) {

p1.y += (win\_min.x - p1.x) \* m;

p1.x = win\_min.x;

} else if (code1 & WIN\_RIGHT\_BIT) {

p1.y += (win\_max.x - p1.x) \* m;

p1.x = win\_max.x;

} else if (code1 & WIN\_BOTTOM\_BIT) {

if (p1.x != p2.x)

p1.x += (win\_min.y - p1.y) / m;

p1.y = win\_min.y;

} else if (code1 & WIN\_TOP\_BIT) {

if (p1.x != p2.x)

p1.x += (win\_max.y - p1.y) / m;

p1.y = win\_max.y;

}

}

}

if (plot\_line) {

glColor3f(1, 0, 0);

glLineWidth(2);

glBegin(GL\_LINES);

glVertex2i(round(p1.x), round(p1.y));

glVertex2i(round(p2.x), round(p2.y));

glEnd();

glFlush();

}

}

void draw\_window(Point win\_min, Point win\_max) {

glColor3f(0, 0, 0);

glBegin(GL\_LINE\_LOOP);

glVertex2i(round(win\_min.x), round(win\_min.y));

glVertex2i(round(win\_min.x), round(win\_max.y));

glVertex2i(round(win\_max.x), round(win\_max.y));

glVertex2i(round(win\_max.x), round(win\_min.y));

glEnd();

glFlush();

}

void init\_clip() {

glClear(GL\_COLOR\_BUFFER\_BIT);

Point win\_min = {60, 60};

Point win\_max = {470, 290};

draw\_window(win\_min, win\_max);

Point p1 = {50, 50};

Point p2 = {490, 310};

glColor3f(0, 0, 1);

glBegin(GL\_LINES);

glVertex2i(round(p1.x), round(p1.y));

glVertex2i(round(p2.x), round(p2.y));

glEnd();

line\_clip(p1, p2, win\_min, win\_max);

}

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

init\_graph(argc, argv);

glutDisplayFunc(init\_clip);

close\_graph();

return EXIT\_SUCCESS;

}

**Variant 3:**

#include<stdio.h>

#include<GL/glut.h>

float xmin=50,ymin=50,xmax=100,ymax=100;

float xvmin=200,yvmin=200,xvmax=400,yvmax=400;

int RIGHT=8,LEFT=2,TOP=4,BOTTOM=1;

float sx,sy,vx1,vy1,vx2,vy2;

float X1,Y1,X2,Y2;

int compute(float x,float y)

{

int code=0;

if(y > ymax)

code = TOP;

else if(y < ymin)

code = BOTTOM;

if(x > xmax)

code = RIGHT;

else if(x < xmin)

code = LEFT;

return code;

}

void cohen(float X1,float Y1,float X2,float Y2)

{

float x,y;

int accept=0,done=0,code\_p,code\_q,code;

code\_p=compute(X1,Y1);

code\_q=compute(X2,Y2);

do

{

if(!(code\_p | code\_q))

{

accept=1;

done=1;

}

else if(code\_p & code\_q)

done=1;

else

{

code=code\_p ? code\_p : code\_q;

if(code & TOP)

{

x=X1+(X2-X1)\*(ymax-Y1)/(Y2-Y1);

y=ymax;

}

else if(code & BOTTOM)

{

x=X1+(X2-X1)\*(ymin-Y1)/(Y2-Y1);

y=ymin;

}

else if(code & RIGHT)

{

y=Y1+(Y2-Y1)\*(xmax-X1)/(X2-X1);

x=xmax;

}

else

{

y=Y1+(Y2-Y1)\*(xmin-X1)/(X2-X1);

x=xmin;

}

if(code==code\_p)

{

X1=x;

Y1=y;

code\_p=compute(X1,Y1);

}

else

{

X2=x;

Y2=y;

code\_q=compute(X2,Y2);

}

}

}while(!done);

if(accept)

{

sx=(xvmax-xvmin)/(xmax-xmin);

sy=(yvmax-yvmin)/(ymax-ymin);

vx1=xvmin+(X1-xmin)\*sx;

vy1=xvmin+(Y1-ymin)\*sy;

vx2=xvmin+(X2-xmin)\*sx;

vy2=xvmin+(Y2-ymin)\*sy;

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1,1,1);

glLineWidth(2);

// The below code is used to draw a enterd lines

glBegin(GL\_LINES);

glVertex2d(X1,Y1);

glVertex2d(X2,Y2);

glEnd();

glColor3f(1,1,1);

// The below code is used to draw a window.

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin,ymin);

glVertex2f(xmax,ymin);

glVertex2f(xmax,ymax);

glVertex2f(xmin,ymax);

glEnd();

cohen(X1,Y1,X2,Y2);

glColor3f(1,1,1);

// The below code is for the view port

glBegin(GL\_LINE\_LOOP);

glVertex2f(xvmin,yvmin);

glVertex2f(xvmax,yvmin);

glVertex2f(xvmax,yvmax);

glVertex2f(xvmin,yvmax);

glEnd();

glColor3f(1,1,1);

// The clipped coordinates at the viewport.

glBegin(GL\_LINES);

glVertex2d(vx1,vy1);

glVertex2d(vx2,vy2);

glEnd();

glFlush();

}

void myinit()

{

glClearColor(0,0,0,1);

gluOrtho2D(0,500,0,500);

}

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

{

printf("\n Enter the points\n");

scanf("%f%f%f%f",&X1,&Y1,&X2,&Y2);

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500,500);

glutCreateWindow("cohen sutherland: C Tathva");

glutDisplayFunc(display);

myinit();

glutMainLoop();

}

**Variant 4:**

#include <stdio.h>

#include <GL/glut.h>

double xmin = 50, ymin = 50, xmax = 100, ymax = 100; //window coordinates

double xvmin = 200, yvmin = 200, xvmax = 300, yvmax = 300; //viewport coordinates

const int LEFT = 1; // code words for LEFT, RIGHT, BOTTOM &TOP.

const int RIGHT = 2;

const int BOTTOM = 4;

const int TOP = 8;

int ComputeOutCode (double x, double y)

{

int code = 0;

if (y > ymax) //above the clip window

code |= TOP;

else if (y < ymin) //below the clip window

code |= BOTTOM;

if (x > xmax) //to the right of clip window

code |= RIGHT;

else if (x < xmin) //to the left of clip window

code |= LEFT;

return code;

}

void CohenSutherland(double x0, double y0,double x1, double y1)

{

int outcode0, outcode1, outcodeOut;

bool accept = false, done = false;

outcode0 = ComputeOutCode (x0, y0); //calculate the region of 1st point

outcode1 = ComputeOutCode (x1, y1); //calculate the region of 2nd point

do

{

if (!(outcode0 | outcode1))

{

accept = true;

done = true;

}

else if (outcode0 & outcode1)

done = true;

else

{

double x, y;

double m = (y1 - y0)/(x1 - x0);

outcodeOut = outcode0? outcode0: outcode1;

if (outcodeOut & TOP)

{

x = x0 + (1/m) \* (ymax - y0);

y = ymax;

}

else if (outcodeOut & BOTTOM)

{

x = x0 + (1/m) \* (ymin - y0);

y = ymin;

}

else if (outcodeOut & RIGHT)

{

y = y0 + m \* (xmax - x0);

x = xmax;

}

else

{

y = y0 + m \* (xmin - x0);

x = xmin;

}

/\* Intersection calculations over \*/

if (outcodeOut == outcode0)

{

x0 = x;

y0 = y;

outcode0 = ComputeOutCode (x0, y0);

}

else

{

x1 = x;

y1 = y;

outcode1 = ComputeOutCode (x1, y1);

}

}

}

while (!done);

if (accept)

{

double sx = (xvmax - xvmin) / (xmax - xmin);

double sy = (yvmax - yvmin) / (ymax - ymin);

double vx0 = xvmin + (x0 - xmin) \* sx;

double vy0 = yvmin + (y0 - ymin) \* sy;

double vx1 = xvmin + (x1 - xmin) \* sx;

double vy1 = yvmin + (y1 - ymin) \* sy;

glBegin(GL\_LINE\_LOOP);

glVertex2f(xvmin, yvmin);

glVertex2f(xvmax, yvmin);

glVertex2f(xvmax, yvmax);

glVertex2f(xvmin, yvmax);

glEnd();

glBegin(GL\_LINES);

glVertex2d (vx0, vy0);

glVertex2d (vx1, vy1);

glEnd();

}

}

void display()

{

double x0 = 60, y0 = 20, x1 = 80, y1 = 120;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1, 1, 1);//white

glBegin(GL\_LINES);

glVertex2d (x0, y0);

glVertex2d (x1, y1);

glEnd();

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin, ymin);

glVertex2f(xmax, ymin);

glVertex2f(xmax, ymax);

glVertex2f(xmin, ymax);

glEnd();

CohenSutherland(x0, y0, x1, y1);

glFlush();

}

void myinit()

{

glClearColor(0, 0, 0, 1);//black

gluOrtho2D(0, 500, 0, 500);

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("Cohen Sutherland Line Clipping Algorithm");

myinit();

glutDisplayFunc(display);

glutMainLoop();

}

1. **To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene**

**Variant 1:**

#include<GL/glut.h>

void teapot(GLfloat x, GLfloat y, GLfloat z)  
{  
glPushMatrix ();  
glTranslatef (x, y, z);  
glutSolidTeapot (0.1);  
glPopMatrix ();  
}

void tableTop(GLfloat x, GLfloat y, GLfloat z)  
{  
glPushMatrix ();  
glTranslatef (x, y, z);  
glScalef (0.6, 0.02, 0.5);  
glutSolidCube (1);  
glPopMatrix ();  
}

void tableLeg(GLfloat x, GLfloat y, GLfloat z)  
{  
glPushMatrix ();  
glTranslatef (x, y, z);  
glScalef (0.02, 0.3, 0.02);  
glutSolidCube (1);  
glPopMatrix ();  
}

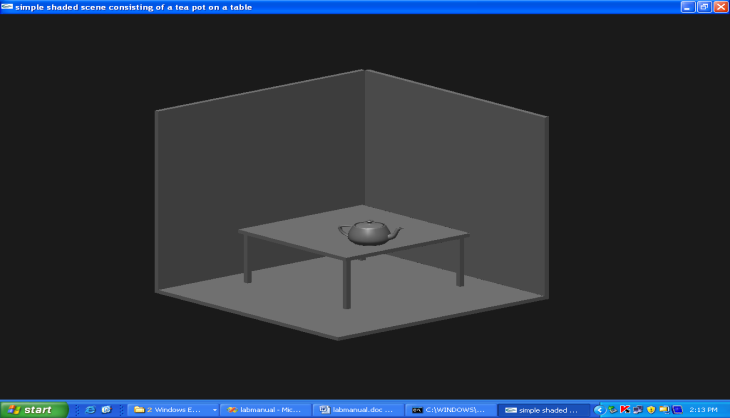
void wall(GLfloat x, GLfloat y, GLfloat z)  
{  
glPushMatrix ();  
glTranslatef (x, y, z);  
glScalef (1, 1, 0.02);  
glutSolidCube (1);  
glPopMatrix ();  
}

void light()  
{  
GLfloat mat\_ambient[] = {1, 1, 1, 1};  
GLfloat mat\_diffuse[] = {0.5, 0.5, 0.5, 1};  
GLfloat mat\_specular[] = {1, 1, 1, 1};  
GLfloat mat\_shininess[] = {50.0f};  
glMaterialfv (GL\_FRONT, GL\_AMBIENT, mat\_ambient);  
glMaterialfv (GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);  
glMaterialfv (GL\_FRONT, GL\_SPECULAR, mat\_specular);  
glMaterialfv (GL\_FRONT, GL\_SHININESS, mat\_shininess);  
GLfloat light\_position[] = {2, 6, 3, 1};  
GLfloat light\_intensity[] = {0.7, 0.7, 0.7, 1};  
glLightfv (GL\_LIGHT0, GL\_POSITION, light\_position);  
glLightfv (GL\_LIGHT0, GL\_DIFFUSE, light\_intensity);  
}

void display()  
{  
GLfloat teapotP = -0.07, tabletopP = -0.15, tablelegP = 0.2, wallP = 0.5;  
glClear (GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);  
glLoadIdentity();  
gluLookAt (-2, 2, 5, 0, 0, 0, 0, 1, 0);  
light ();  
teapot (0, teapotP, 0);  
tableTop (0, tabletopP, 0);  
tableLeg (tablelegP, -0.3, tablelegP);  
tableLeg (-tablelegP, -0.3, tablelegP);  
tableLeg (-tablelegP, -0.3, -tablelegP);  
tableLeg (tablelegP, -0.3, -tablelegP);  
wall (0, 0, -wallP);  
glRotatef (90, 1, 0, 0);  
wall (0, 0, wallP);  
glRotatef (90, 0, 1, 0);  
wall (0, 0, wallP);  
glFlush ();  
}  
void init()  
{  
glClearColor (0, 0, 0, 1);  
glMatrixMode (GL\_PROJECTION);  
glLoadIdentity ();  
glOrtho (-1, 1, -1, 1, -1, 10);  
glMatrixMode (GL\_MODELVIEW);  
}

int main (int argc, char \*\*argv)  
{  
glutInit(&argc, argv);  
glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH);  
glutInitWindowSize(500, 500);  
glutInitWindowPosition(0, 0);  
glutCreateWindow("Teapot on a table");  
init();  
glutDisplayFunc(display);  
glEnable(GL\_LIGHTING);  
glEnable(GL\_LIGHT0);  
glShadeModel(GL\_SMOOTH);  
glEnable(GL\_NORMALIZE);  
glEnable(GL\_DEPTH\_TEST);  
glutMainLoop();  
return 0;  
}

**Output:**

****

**Variant 2:**

**#include<stdio.h>**

**#include<GL/glut.h>**

**void** **wall**(**double** thickness)

{

glPushMatrix();

glTranslated(0.5,0.5\*thickness,0.5);

glScaled(1.0,thickness,1.0);

glutSolidCube(1.0);

glPopMatrix();

}

**void** **tableLeg**(**double** thick,**double** len)

{

glPushMatrix();

glTranslated(0,len/2,0);

glScaled(thick,len,thick);

glutSolidCube(1.0);

glPopMatrix();

}

**void** **table**(**double** topWid,**double** topThick,**double** legThick,**double** legLen)

{

glPushMatrix();

glTranslated(0,legLen,0);

glScaled(topWid,topThick,topWid);

glutSolidCube(1.0);

glPopMatrix();

**double** dist=0.95\*topWid/2.0-legThick/2.0;

glPushMatrix();

glTranslated(dist,0,dist);

tableLeg(legThick,legLen);

glTranslated(0.0,0.0,-2\*dist);

tableLeg(legThick,legLen);

glTranslated(-2\*dist,0,2\*dist);

tableLeg(legThick,legLen);

glTranslated(0,0,-2\*dist);

tableLeg(legThick,legLen);

glPopMatrix();

}

**void** **displaySolid**(**void**)

{

GLfloat mat\_ambient[]={0.7f,0.7f,0.7f,1.0f};

GLfloat mat\_diffuse[]={0.5f,0.5f,0.5f,1.0f};

GLfloat mat\_specular[]={1.0f,1.0f,1.0f,1.0f};

GLfloat mat\_shininess[]={50.0f};

*//The glMaterialfv function specifies material parameters for the lighting model.*

glMaterialfv(GL\_FRONT,GL\_AMBIENT,mat\_ambient);

glMaterialfv(GL\_FRONT,GL\_DIFFUSE,mat\_diffuse);

glMaterialfv(GL\_FRONT,GL\_SPECULAR,mat\_specular);

glMaterialfv(GL\_FRONT,GL\_SHININESS,mat\_shininess);

GLfloat lightIntensity[]={0.7f,0.7f,0.7f,0.7f};

GLfloat light\_position[]={2.0f,6.0f,3.0f,0.0f};

*//The glLightfv function returns light source parameter values.*

glLightfv(GL\_LIGHT0,GL\_POSITION,light\_position);

glLightfv(GL\_LIGHT0,GL\_DIFFUSE,lightIntensity);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

**double** winHt=1.0;

glOrtho(-winHt\*64/48.0,winHt\*64/48.0,-winHt,winHt,0.1,100.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(2.3,1.3,2.0,0.0,0.25,0.0,0.0,1.0,0.0);

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

glPushMatrix();

glTranslated(0.4,0.4,0.6);

glRotated(45,0,0,1);

glScaled(0.08,0.08,0.08);

glPopMatrix();

glPushMatrix();

glTranslated(0.6,0.38,0.5);

glRotated(30,0,1,0);

glutSolidTeapot(0.08);

glPopMatrix();

glPushMatrix();

glTranslated(0.25,0.42,0.35);

glPopMatrix();

glPushMatrix();

glTranslated(0.4,0,0.4);

table(0.6,0.02,0.02,0.3);

glPopMatrix();

wall(0.02);

glPushMatrix();

glRotated(90.0,0.0,0.0,1.0); *//draw second wall after rotating x axis by 90degre*

wall(0.02);

glPopMatrix();

glPushMatrix();

glRotated(-90.0,1.0,0.0,0.0); *//draw floor*

wall(0.02);

glPopMatrix();

glFlush();

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(640,480);

glutInitWindowPosition(100,100);

glutCreateWindow("Simple shaded scene consisting of a teapot");

glutDisplayFunc(displaySolid);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glShadeModel(GL\_SMOOTH);*//Specifies a symbolic value representing a shading technique. Accepted values are GL\_FLAT and GL\_SMOOTH.*

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_NORMALIZE);

glClearColor(0.1,0.1,0.1,0.0);

glViewport(0,0,640,480);

glutMainLoop();

}

**Variant 3:**

#include<GL/glut.h>

void displaySolid(void)

{

            glClearColor(0.5,0.5,0.1,0.0);

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

            glMatrixMode(GL\_PROJECTION);

            glLoadIdentity();

            glOrtho(-100,100,-100,100,-100,100);

            glMatrixMode(GL\_MODELVIEW);

            glLoadIdentity();

            //set properties of the surface material

            GLfloat mat\_ambient[]={0.0f,1.0f,1.0f,1.0f};

            GLfloat mat\_diffuse[]={1.0f,0.5f,1.0f,1.0f};

            GLfloat mat\_specular[]={0.5f,0.5f,1.0f,1.0f};

            GLfloat mat\_shininess[]={25.0f};

            glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_AMBIENT,mat\_ambient);

            glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_DIFFUSE,mat\_diffuse);

            glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_SPECULAR,mat\_specular);

            glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_SHININESS,mat\_shininess);

            //set the light source properties

            GLfloat lightIntensity[]={1.0f,0.7f,0.7f,1.0f};

            GLfloat light\_position[]={25.0f,50.0f,50.0f,1.0f};

            glLightfv(GL\_LIGHT0, GL\_POSITION,light\_position);

            glLightfv(GL\_LIGHT0,GL\_DIFFUSE,lightIntensity);

            glPushMatrix();

            glTranslated(0,30,0);

            glRotatef(35,1,0.5,0);

            //glScaled(1,8,1);

            glutSolidTeapot(10);

            //glutWireTeapot(10);

            glPopMatrix();

            GLfloat mat\_ambient1[]={1.0f,0.0f,0.0f,1.0f};

            GLfloat mat\_diffuse1[]={1.0f,1.0f,0.0f,1.0f};

            GLfloat mat\_specular1[]={1.0f,1.0f,0.5f,1.0f};

            GLfloat mat\_shininess1[]={25.0f};

            glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_AMBIENT,mat\_ambient1);

            glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_DIFFUSE,mat\_diffuse1);

            glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_SPECULAR,mat\_specular1);

            glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_SHININESS,mat\_shininess1);

            //set the light source properties

            GLfloat lightIntensity1[]={0.5f,0.5f,0.5f,1.0f};

            GLfloat light\_position1[]={25.0f,50.0f,50.0f,1.0f};

            glLightfv(GL\_LIGHT0, GL\_POSITION,light\_position1);

            glLightfv(GL\_LIGHT0,GL\_DIFFUSE,lightIntensity1);

            //top surface

            glPushMatrix();

            glTranslated(0,20,0);

            glRotatef(-80,1,0.5,0.8);

            //glRotatef(,0,0,1);

            glScalef(1.5,1.5,0.1);

            glutSolidCube(50);

            glPopMatrix();

            //First Leg

            glPushMatrix();

            glTranslated(-45,-10,-5);

            glRotatef(45,0,1,0);

            glScalef(0.4,5.5,0.4);

            glutSolidCube(10);

            glPopMatrix();

            //Second Leg

            glPushMatrix();

            glTranslated(-10,-25,5);

            glRotatef(45,0,1,0);

            glScalef(0.4,4.5,0.4);

            glutSolidCube(10);

            glPopMatrix();

            //Third Leg

            glPushMatrix();

            glTranslated(45,-5,-10);

            glRotatef(45,0,1,0);

            glScaled(0.4,5.5,0.4);

            glutSolidCube(10);

            glPopMatrix();

            //Fourth Leg

            glPushMatrix();

            glTranslated(10,5,-35);

            glRotatef(45,0,1,0);

            glScalef(0.4,6,0.4);

            glutSolidCube(10);

            glPopMatrix();

            glFlush();

}

void main(int argc, char \*argv[])

{

            glutInit(&argc,argv);

            glutInitWindowSize(600,600);

            glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH);

            glutInitWindowPosition(10,10);

            glutInitWindowSize(500,500);

            glutCreateWindow("Simple shaded scene consisting of a tea pot on a table");

            glutDisplayFunc(displaySolid);

            glEnable(GL\_LIGHTING); // enable Lighting

            glEnable(GL\_LIGHT0);  // enable the light source

            glShadeModel(GL\_SMOOTH); // set the shading model GL\_FLAT or GL\_SMOOTH

            glEnable(GL\_DEPTH\_TEST);

            glEnable(GL\_NORMALIZE);

            glutMainLoop();

}

**Variant 4:**

#include<GL/glut.h>

#include<stdio.h>

**void wall(double thickness)**

**{**

glPushMatrix();

glTranslated(0.5,0.5\*thickness, 0.5);

glScaled(1.0,thickness, 1.0);

glutSolidCube(1.0);

glPopMatrix();

**}**

**void tableleg(double thick, double len)**

**{**

glPushMatrix();

glTranslated(0,len/2,0);

glScaled(thick, len, thick);

glutSolidCube(1.0);

glPopMatrix();

**}**

**void table(double topwid, double topthick, double legthick, double leglen)**

**{**

glPushMatrix();

glTranslated(0,leglen,0);

glScaled(topwid, topthick, topwid);

glutSolidCube(1.0);

glPopMatrix();

double dist=0.95\*topwid/2.0-legthick/2.0;

glPushMatrix();

glTranslated(dist, 0, dist);

tableleg(legthick, leglen);

glTranslated(0.0,0.0,-2\*dist);

tableleg(legthick, leglen);

glTranslated(-2\*dist, 0, 2\*dist);

tableleg(legthick, leglen);

glTranslated(0,0,-2\*dist);

tableleg(legthick,leglen);

glPopMatrix();

**}**

**void displaySolid(void)**

**{**

GLfloat mat\_ambient[]={0.7f,0.7f,0.7f,1.0f};

GLfloat mat\_diffuse[]={0.5f,0.5f,0.5f,1.0f};

GLfloat mat\_specular[]={1.0f,1.0f,1.0f,1.0f};

GLfloat mat\_shininess[]={50.0f};

glMaterialfv(GL\_FRONT,GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT,GL\_DIFFUSE, mat\_diffuse);

glMaterialfv(GL\_FRONT,GL\_SPECULAR, mat\_specular);

glMaterialfv(GL\_FRONT,GL\_SHININESS, mat\_shininess);

GLfloat lightintensity[]={0.7f,0.7f,0.7f,1.0f};

GLfloat lightposition[]={2.0f,6.0f,3.0f,0.0f};

glLightfv(GL\_LIGHT0, GL\_POSITION, lightposition);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightintensity);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

double winht=1.0;

glOrtho(-winht\*64/48, winht\*64/48, -winht, winht,

0.1, 100.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(2.3,1.3,2.0,0.0,0.25,0.0,0.0,1.0,0.0);

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

glPushMatrix();

glTranslated(0.6,0.38,0.5);

glRotated(30,0,1,0);

glutSolidTeapot(0.08);

glPopMatrix();

glPushMatrix();

glTranslated(0.4,0,0.4);

table(0.6,0.02,0.02,0.3);

glPopMatrix();

wall(0.02);

glPushMatrix();

glRotated(90.0,0.0,0.0,1.0);

wall(0.02);

glPopMatrix();

glPushMatrix();

glRotated(-90.0,1.0,0.0,0.0);

wall(0.02);

glPopMatrix();

glFlush();

**}**

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

**{**

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowPosition(50,50);

glutInitWindowSize(400,300);

glutCreateWindow("Shaded Scene");

glutDisplayFunc(displaySolid);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glShadeModel(GL\_SMOOTH);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_NORMALIZE);

glClearColor(0.1,0.1,0.1,0.0);

glViewport(0,0,640,480);

glutMainLoop();

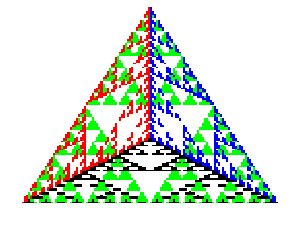
1. **Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user**

**Variant 1:**

#include<stdlib.h>  
#include<stdio.h>  
#include<GL/glut.h>  
typedef float point[3];  
point v[]={{0.0,0.0,1.0},{0.0,1.0,0.0},{-1.0,-0.5,0.0}, {1.0,-0.5,0.0}};  
int n;

void triangle(point a,point b,point c)  
{  
glBegin(GL\_POLYGON);  
glVertex3fv(a);  
glVertex3fv(b);  
glVertex3fv(c);  
glEnd();  
}  
void divide\_triangle(point a,point b,point c,int m)  
{  
point v1,v2,v3;  
int j;  
if(m>0)  
{  
for(j=0;j<3;j++)  
v1[j]=(a[j]+b[j])/2;  
for(j=0;j<3;j++)  
v2[j]=(a[j]+c[j])/2;  
for(j=0;j<3;j++)  
v3[j]=(c[j]+b[j])/2;  
divide\_triangle(a,v1,v2,m-1);  
divide\_triangle(c,v2,v3,m-1);  
divide\_triangle(b,v3,v1,m-1);  
}  
else(triangle(a,b,c));  
}  
void tetrahedron(int m)  
{  
glColor3f(1.0,0.0,0.0);  
divide\_triangle(v[0],v[1],v[2],m);  
glColor3f(0.0,1.0,0.0);  
divide\_triangle(v[3],v[2],v[1],m);  
glColor3f(0.0,0.0,1.0);  
divide\_triangle(v[0],v[3],v[1],m);  
glColor3f(0.0,0.0,0.0);  
divide\_triangle(v[0],v[2],v[3],m);  
}  
void display(void)  
{  
glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);  
glLoadIdentity();  
tetrahedron(n);  
glFlush();  
}  
void myReshape(int w,int h)  
{  
glViewport(0,0,w,h);  
glMatrixMode(GL\_PROJECTION);  
glLoadIdentity();  
if(w<=h)  
glOrtho(-2.0,2.0,-2.0\*(GLfloat)h/(GLfloat)w,  
2.0\*(GLfloat)h/(GLfloat)w,-10.0,10.0);  
else  
glOrtho(-2.0\*(GLfloat)w/(GLfloat)h,  
2.0\*(GLfloat)w/(GLfloat)h,-2.0,2.0,-10.0,10.0);  
glMatrixMode(GL\_MODELVIEW);  
glutPostRedisplay();  
}  
int main(int argc,char \*\* argv)  
{  
printf("No of Division?: ");  
scanf("%d",&n);  
glutInit(&argc,argv);  
glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH);  
glutCreateWindow("3D gasket");  
glutReshapeFunc(myReshape);  
glutDisplayFunc(display);  
glEnable(GL\_DEPTH\_TEST);  
glClearColor(1.0,1.0,1.0,0.0);  
glutMainLoop();  
return 0;  
}

**Output**

****

**Variant 2:**

|  |
| --- |
| #include <windows.h> |
|  | #include <GL/glut.h> |
|  |  |
|  | GLfloat d=0; |
|  |  |
|  | void spin() |
|  | { |
|  | d=d+0.005; |
|  | if(d>360) |
|  | d=0; |
|  | glutPostRedisplay(); |
|  | } |
|  | void tri(GLfloat a[3],GLfloat b[3],GLfloat c[3]) |
|  | { |
|  | glBegin(GL\_TRIANGLES); |
|  | glVertex3fv(a); |
|  | glVertex3fv(b); |
|  | glVertex3fv(c); |
|  | glEnd(); |
|  | } |
|  | void tetra(GLfloat a[3],GLfloat b[3],GLfloat c[3],GLfloat d[3]) |
|  | { |
|  | glColor3f(0,0,0); |
|  | tri(a,b,c); |
|  | glColor3f(1,0,0); |
|  | tri(a,b,d); |
|  | glColor3f(0,1,0); |
|  | tri(b,c,d); |
|  | glColor3f(0,0,1); |
|  | tri(a,c,d); |
|  | } |
|  |  |
|  | void div(GLfloat p0[3],GLfloat p1[3],GLfloat p2[3],GLfloat p3[3],int n) |
|  | { |
|  | GLfloat p01[3],p12[3],p20[3],p03[3],p13[3],p23[3]; |
|  | if(n==0) |
|  | tetra(p0,p1,p2,p3); |
|  | else |
|  | { |
|  | p01[0]=(p0[0]+p1[0])/2; |
|  | p01[1]=(p0[1]+p1[1])/2; |
|  | p01[2]=(p0[2]+p1[2])/2; |
|  |  |
|  | p12[0]=(p1[0]+p2[0])/2; |
|  | p12[1]=(p1[1]+p2[1])/2; |
|  | p12[2]=(p1[2]+p2[2])/2; |
|  |  |
|  | p20[0]=(p2[0]+p0[0])/2; |
|  | p20[1]=(p2[1]+p0[1])/2; |
|  | p20[2]=(p2[2]+p0[2])/2; |
|  |  |
|  | p03[0]=(p0[0]+p3[0])/2; |
|  | p03[1]=(p0[1]+p3[1])/2; |
|  | p03[2]=(p0[2]+p3[2])/2; |
|  |  |
|  | p13[0]=(p1[0]+p3[0])/2; |
|  | p13[1]=(p1[1]+p3[1])/2; |
|  | p13[2]=(p1[2]+p3[2])/2; |
|  |  |
|  | p23[0]=(p2[0]+p3[0])/2; |
|  | p23[1]=(p2[1]+p3[1])/2; |
|  | p23[2]=(p2[2]+p3[2])/2; |
|  |  |
|  |  |
|  | div(p0,p01,p20,p03,n-1); |
|  | div(p01,p1,p12,p13,n-1); |
|  | div(p12,p2,p20,p23,n-1); |
|  | div(p03,p13,p23,p3,n-1); |
|  | } |
|  | } |
|  |  |
|  | void draw() |
|  | { |
|  | GLfloat p[4][3]={ {-0.5,-0.5,0.5},{0.5,-0.5,0.5},{0,0.5,0.5},{0,0,-0.5} }; |
|  | glClearColor(1,1,1,1); |
|  | glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); |
|  | glLoadIdentity(); |
|  | glRotatef(d,0,1,0); |
|  | div(p[0],p[1],p[2],p[3],3); |
|  |  |
|  | glutSwapBuffers(); |
|  |  |
|  | } |
|  | int main(int c,char \*v[]) |
|  | { |
|  | glutInit(&c,v); |
|  | glutInitWindowPosition(200,150); |
|  | glutInitWindowSize(648,480); |
|  | glutInitDisplayMode(GLUT\_RGB | GLUT\_DEPTH | GLUT\_DOUBLE); |
|  | glutCreateWindow("Sierpinski Gasket"); |
|  | glutDisplayFunc(draw); |
|  | glutIdleFunc(spin); |
|  | glEnable(GL\_DEPTH\_TEST); |
|  | glutMainLoop(); |
|  | return 0; |
|  | } |

**Variant 3:**

#include <stdlib.h>

#include <stdio.h>

#include <GL/glut.h>

typedef float point**[**3**]**;

point v**[]**=**{** **{**0.0, 0.0, 1.0**}**,

**{**0.0, 0.942809, -0.33333**}**,

**{**-0.816497, -0.471405, -0.333333**}**,

**{**0.816497, -0.471405, -0.333333**}}**;

static GLfloat theta**[]** = **{**0.0,0.0,0.0**}**;

int n;

**void** triangle**(** point a, point b, point c**)**

**{**

glBegin**(**GL\_POLYGON**)**;

glNormal3fv**(**a**)**;

glVertex3fv**(**a**)**;

glVertex3fv**(**b**)**;

glVertex3fv**(**c**)**;

glEnd**()**;

**}**

**void** divide\_triangle**(**point a, point b, point c, int m**)**

**{**

point v1, v2, v3;

int j;

**if(**m**>**0**)**

**{**

**for(**j=0; j**<**3; j++**)**

v1**[**j**]**=**(**a**[**j**]**+b**[**j**])**/2;

**for(**j=0; j**<**3; j++**)**

v2**[**j**]**=**(**a**[**j**]**+c**[**j**])**/2;

**for(**j=0; j**<**3; j++**)**

v3**[**j**]**=**(**b**[**j**]**+c**[**j**])**/2;

divide\_triangle**(**a, v1, v2, m-1**)**;

divide\_triangle**(**c, v2, v3, m-1**)**;

divide\_triangle**(**b, v3, v1, m-1**)**;

**}**

**else(**triangle**(**a,b,c**))**;

**}**

**void** tetrahedron**(** int m**)**

**{**

glColor3f**(**1.0,0.0,0.0**)**;

divide\_triangle**(**v**[**0**]**, v**[**1**]**, v**[**2**]**, m**)**;

glColor3f**(**0.0,1.0,0.0**)**;

divide\_triangle**(**v**[**3**]**, v**[**2**]**, v**[**1**]**, m**)**;

glColor3f**(**0.0,0.0,1.0**)**;

divide\_triangle**(**v**[**0**]**, v**[**3**]**, v**[**1**]**, m**)**;

glColor3f**(**0.0,0.0,0.0**)**;

divide\_triangle**(**v**[**0**]**, v**[**2**]**, v**[**3**]**, m**)**;

**}**

**void** display**(void)**

**{**

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

glLoadIdentity**()**;

tetrahedron**(**n**)**;

glFlush**()**;

**}**

**void** myReshape**(**int w, int h**)**

**{**

glViewport**(**0, 0, w, h**)**;

glMatrixMode**(**GL\_PROJECTION**)**;

glLoadIdentity**()**;

**if** **(**w **<**= h**)**

glOrtho**(**-2.0, 2.0, -2.0 **\*** **(**GLfloat**)** h / **(**GLfloat**)** w, 2.0 **\*** **(**GLfloat**)** h / **(**GLfloat**)** w, -10.0, 10.0**)**;

**else**

glOrtho**(**-2.0 **\*** **(**GLfloat**)** w / **(**GLfloat**)** h, 2.0 **\*** **(**GLfloat**)** w / **(**GLfloat**)** h, -2.0, 2.0, -10.0, 10.0**)**;

glMatrixMode**(**GL\_MODELVIEW**)**;

glutPostRedisplay**()**;

**}**

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

**{**

printf**(**" No. of Divisions ? "**)**;

scanf**(**"%d",&n**)**;

glutInit**(**&argc, argv**)**;

glutInitDisplayMode**(**GLUT\_SINGLE **|** GLUT\_RGB **|** GLUT\_DEPTH**)**;

glutInitWindowSize**(**500, 500**)**;

glutCreateWindow**(**"3D Gasket"**)**;

glutReshapeFunc**(**myReshape**)**;

glutDisplayFunc**(**display**)**;

glEnable**(**GL\_DEPTH\_TEST**)**;

glClearColor **(**1.0, 1.0, 1.0, 1.0**)**;

glutMainLoop**()**;

**}**

**Variant 4:**

#include<stdio.h>

#include<math.h>

#include<iostream>

#include<GL/glut.h>

using namespace std;

float v[4][3] = { { 0.0,0.0,1.0 },{ 0,1,-1 },{ -0.8,-0.4,-1 },{ 0.8,-0.4,-1 } };

int n;

void triangle(float a[], float b[], float c[])

{

glBegin(GL\_POLYGON);

glVertex3fv(a);

glVertex3fv(b);

glVertex3fv(c);

glEnd();

}

void divide\_triangle(float a[], float b[], float c[], int m)

{

float v1[3], v2[3], v3[3];

int i;

if (m>0)

{

for (i = 0; i<3; i++) v1[i] = (a[i] + b[i]) / 2;

for (i = 0; i<3; i++) v2[i] = (a[i] + c[i]) / 2;

for (i = 0; i<3; i++) v3[i] = (b[i] + c[i]) / 2;

divide\_triangle(a, v1, v2, m - 1);

divide\_triangle(c, v2, v3, m - 1);

divide\_triangle(b, v3, v1, m - 1);

}

else (triangle(a, b, c));

}

void tetrahedron(int m)

{

glColor3f(1.0, 0.0, 0.0);

divide\_triangle(v[0], v[1], v[2], m);

glColor3f(0.0, 1.0, 0.0);

divide\_triangle(v[3], v[2], v[1], m);

glColor3f(0.0, 0.0, 1.0);

divide\_triangle(v[0], v[3], v[1], m);

glColor3f(1.0, 1.0, 0.0);

divide\_triangle(v[0], v[2], v[3], m);

}

void display()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-2.0, 2.0, -2.0, 2.0, -10.0, 10.0);

glMatrixMode(GL\_MODELVIEW);

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

tetrahedron(n);

glFlush();

glutPostRedisplay();

}

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

{

cout << "Enter the number of divisions: ";

cin >> n;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("3D Gasket");

glutDisplayFunc(display);

glEnable(GL\_DEPTH\_TEST);

glClearColor(1.0, 1.0, 1.0, 1.0);

glutMainLoop();

return 0;

}

1. Develop a menu driven program to animate a flag using Bezier Curve algorithm

**Variant 1**

#include<GL/glut.h>  
#include<stdio.h>  
#include<math.h>  
#define PI 3.1416  
float theta = 0;

struct point  
{  
GLfloat x, y, z;  
};

int factorial (int n)  
{  
if (n<=1)  
return (1);  
else  
n = n \* factorial ( n-1 );  
return n;  
}

void computeNcR (int n, int \*hold\_ncr\_values)  
{  
int r;  
for (r=0; r<=n; r++)  
{  
hold\_ncr\_values [r] = factorial (n) / ( factorial (n-r) \* factorial (r) );  
}  
}

void computeBezierPoints (float t, point \*actual\_bezier\_point,  
int number\_of\_control\_points,  
point \*control\_points\_array, int \*hold\_ncr\_values)  
{  
int i, n = number\_of\_control\_points - 1;  
float bernstein\_polynomial;  
actual\_bezier\_point -> x = 0;  
actual\_bezier\_point -> y = 0;  
actual\_bezier\_point -> z = 0;  
for ( i=0; i<number\_of\_control\_points; i++ )  
{  
bernstein\_polynomial = hold\_ncr\_values [i] \* pow(t, i) \* pow( 1-t, n-i);  
actual\_bezier\_point->x += bernstein\_polynomial \* control\_points\_array [i].x;  
actual\_bezier\_point->y += bernstein\_polynomial \* control\_points\_array [i].y;  
actual\_bezier\_point->z += bernstein\_polynomial \* control\_points\_array [i].z;  
}  
}

void Bezier (point \*control\_points\_array, int number\_of\_control\_points,  
int number\_of\_bezier\_points)  
{  
point actual\_bezier\_point;  
float t;  
int \*hold\_ncr\_values, i;  
hold\_ncr\_values = new int [number\_of\_control\_points];  
computeNcR (number\_of\_control\_points - 1, hold\_ncr\_values);

glBegin (GL\_LINE\_STRIP);  
for(i=0; i<=number\_of\_bezier\_points; i++)  
{  
t=float (i) / float (number\_of\_bezier\_points);  
computeBezierPoints ( t, &actual\_bezier\_point, number\_of\_control\_points,  
control\_points\_array, hold\_ncr\_values );// 5 parameters  
glVertex2f (actual\_bezier\_point.x, actual\_bezier\_point.y);  
}  
glEnd ();  
delete [] hold\_ncr\_values;  
}

void display()  
{  
glClear (GL\_COLOR\_BUFFER\_BIT);  
int number\_of\_control\_points= 4, number\_of\_bezier\_points= 20;  
point control\_points\_array[4]= {{100, 400, 0}, {150, 450, 0}, {250, 350, 0},{300, 400, 0}};  
control\_points\_array[1].x += 50 \* sin (theta \* PI/180.0);  
control\_points\_array[1].y += 25 \* sin (theta \* PI/180.0);

control\_points\_array[2].x -= 50 \* sin ((theta+30) \* PI/180.0);  
control\_points\_array[2].y -= 50 \* sin ((theta+30) \* PI/180.0);

control\_points\_array[3].x -= 25 \* sin ((theta-30) \* PI/180.0);  
control\_points\_array[3].y += sin ((theta-30) \* PI/180.0);  
theta += 2; //animating speed  
glPushMatrix ();  
glPointSize (5);  
glColor3f (1, 0.4, 0.2);

for (int i=0; i<50; i++)  
{  
glTranslatef(0, -0.8, 0 );  
Bezier(control\_points\_array, number\_of\_control\_points, number\_of\_bezier\_points);  
}

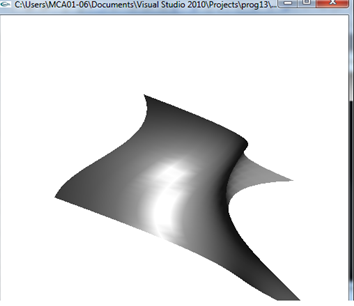
glColor3f(1, 1, 1);  
for(int i=0; i<50; i++)  
{  
glTranslatef(0, -0.8, 0);  
Bezier(control\_points\_array, number\_of\_control\_points, number\_of\_bezier\_points);

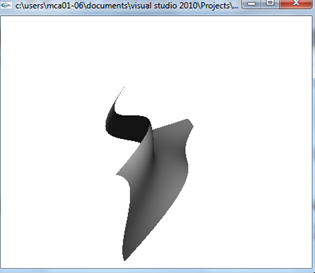
}  
glColor3f(0, 1, 0);  
for(int i=0; i<50; i++)  
{  
glTranslatef(0, -0.8, 0);  
Bezier(control\_points\_array, number\_of\_control\_points, number\_of\_bezier\_points);  
}

glPopMatrix();  
glLineWidth(5);  
glColor3f(0.7, 0.5,0.3);  
glBegin(GL\_LINES);  
glVertex2f(100,400);  
glVertex2f(100,40);  
glEnd();  
glutPostRedisplay();  
glutSwapBuffers();  
}

void init ()  
{  
glMatrixMode(GL\_PROJECTION);  
glLoadIdentity();  
gluOrtho2D(0,500,0,500);  
}

int main(int argc, char \*\* argv)  
{  
glutInit(&argc, argv);  
glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);  
glutInitWindowPosition(0, 0);  
glutInitWindowSize(500,500);  
glutCreateWindow ("Bezier Curve - updated");  
init ();  
glutDisplayFunc (display);  
glutMainLoop ();  
return 0;  
}

****

****

**Variant 2**

//Develop a menu driven program to animate a flag using Bezier Curve algorithm

#define PI 3.1416

GLsizei winWidth = 600,

winHeight = 600; GLfloat xwcMin = 0.0, xwcMax = 130.0; GLfloat ywcMin = 0.0, ywcMax = 130.0; int animate=1;

typedef struct wcPt3D

{ GLfloat x, y, z;

};

void bino(GLint n, GLint \*C)

{ GLint k, j;

for(k=0;k<=n;k++)

{ C[k]=1;

for(j=n;j>=k+1; j--)

C[k]\*=j; for(j=n-k;j>=2;j--)

C[k]/=j; } }

void computeBezPt(GLfloat u, wcPt3D \*bezPt, GLint nCtrlPts, wcPt3D \*ctrlPts, GLint \*C)

{ GLint k, n=nCtrlPts-1; GLfloat bezBlendFcn;

bezPt ->x =bezPt ->y = bezPt->z=0.0;

for(k=0; k< nCtrlPts; k++)

{ bezBlendFcn = C[k] \* pow(u, k) \* pow( 1-u, n-k); bezPt ->x += ctrlPts[k].x \* bezBlendFcn; bezPt ->y += ctrlPts[k].y \* bezBlendFcn; bezPt ->z += ctrlPts[k].z \* bezBlendFcn; } } v

oid bezier(wcPt3D \*ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)

{ wcPt3D bezCurvePt; GLfloat u; GLint \*C, k; C= new GLint[nCtrlPts]; bino(nCtrlPts-1, C);

glBegin(GL\_LINE\_STRIP);

for(k=0; k<=nBezCurvePts; k++)

{ u=GLfloat(k)/GLfloat(nBezCurvePts);

computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);

glVertex2f(bezCurvePt.x, bezCurvePt.y); }

glEnd(); delete[]C; }

void displayFcn()

{ if(animate) { GLint nCtrlPts = 4, nBezCurvePts =20;

static float theta = 0;

wcPt3D ctrlPts[4] = { {20, 100, 0}, {30, 110, 0}, {50, 90, 0}, {60, 100, 0}};

ctrlPts[1].x +=10\*sin(theta \* PI/180.0); ctrlPts[1].y +=5\*sin(theta \* PI/180.0); ctrlPts[2].x -= 10\*sin((theta+30) \* PI/180.0); ctrlPts[2].y -= 10\*sin((theta+30) \* PI/180.0); ctrlPts[3].x-= 4\*sin((theta) \* PI/180.0); ctrlPts[3].y += sin((theta-30) \* PI/180.0); theta+=0.1;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 1.0, 1.0);

glPointSize(5);

glPushMatrix();

glLineWidth(5);

glColor3f(255/255, 153/255.0, 51/255.0); //Indian flag: Orange color code

**Variant 3:**

#include<GL/glut.h>

#include<stdio.h>

#include<math.h>

#define PI 3.1416

GLsizei winWidth = 600, winHeight = 600;

GLfloat xwcMin = 0.0, xwcMax = 130.0; GLfloat ywcMin = 0.0, ywcMax = 130.0; int animate=1; typedef struct wcPt3D

{

GLfloat x, y, z;

};

void bino(GLint n, GLint \*C)

{ GLint k, j; for(k=0;k<=n;k++)

{ C[k]=1;

for(j=n;j>=k+1; j--) C[k]\*=j; for(j=n-k;j>=2;j--)

C[k]/=j;

} }

void computeBezPt(GLfloat u, wcPt3D \*bezPt, GLint nCtrlPts, wcPt3D \*ctrlPts,

GLint

\*C)

{

GLint k, n=nCtrlPts-1; GLfloat bezBlendFcn;

bezPt ->x =bezPt ->y = bezPt->z=0.0; for(k=0; k< nCtrlPts; k++)

{

bezBlendFcn = C[k] \* pow(u, k) \* pow( 1-u, n-k); bezPt ->x += ctrlPts[k].x \* bezBlendFcn; bezPt ->y += ctrlPts[k].y \* bezBlendFcn; bezPt ->z += ctrlPts[k].z \* bezBlendFcn;

} }

void bezier(wcPt3D \*ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)

{

wcPt3D bezCurvePt;

GLfloat u;

GLint \*C, k;

C= new GLint[nCtrlPts]; bino(nCtrlPts-1, C); glBegin(GL\_LINE\_STRIP); for(k=0; k<=nBezCurvePts; k++)

{

u=GLfloat(k)/GLfloat(nBezCurvePts);

computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C); glVertex2f(bezCurvePt.x, bezCurvePt.y);

} glEnd(); delete[]C;

}

void displayFcn()

{

if(animate)

{

GLint nCtrlPts = 4, nBezCurvePts =20; static float theta = 0; wcPt3D ctrlPts[4] = {

{20, 100, 0},

{30, 110, 0},

{50, 90, 0}, {60, 100, 0}};

ctrlPts[1].x +=10\*sin(theta \* PI/180.0); ctrlPts[1].y +=5\*sin(theta \* PI/180.0); ctrlPts[2].x -= 10\*sin((theta+30) \* PI/180.0); ctrlPts[2].y -= 10\*sin((theta+30) \* PI/180.0); ctrlPts[3].x-= 4\*sin((theta) \* PI/180.0); ctrlPts[3].y += sin((theta-30) \* PI/180.0); theta+=0.1;

glClear(GL\_COLOR\_BUFFER\_BIT); glColor3f(1.0, 1.0, 1.0); glPointSize(5); glPushMatrix(); glLineWidth(5);

glColor3f(255/255, 153/255.0, 51/255.0); //Indian flag: Orange color code for(int i=0;i<8;i++)

{

glTranslatef(0, -0.8, 0);

bezier(ctrlPts, nCtrlPts, nBezCurvePts);

}

glColor3f(1, 1, 1); //Indian flag: white color code for(int i=0;i<8;i++)

{

glTranslatef(0, -0.8, 0);

bezier(ctrlPts, nCtrlPts, nBezCurvePts);

}

glColor3f(19/255.0, 136/255.0, 8/255.0); //Indian flag: green color code for(int i=0;i<8;i++)

{

glTranslatef(0, -0.8, 0);

bezier(ctrlPts, nCtrlPts, nBezCurvePts);

} glPopMatrix(); glColor3f(0.7, 0.5,0.3); glLineWidth(5); glBegin(GL\_LINES); glVertex2f(20,100); glVertex2f(20,40); glEnd(); glFlush(); glutPostRedisplay();

glutSwapBuffers();

}

} // Menu exit

void handlemenu(int value)

{

switch (value) {

case 4: exit(0); break;

}

}

//Colors menu void cmenu(int value){

switch(value){ case 1:

animate=1;

glutPostRedisplay();

break; case 2:

animate=0; glutPostRedisplay(); break;

}

}

void winReshapeFun(GLint newWidth, GLint newHeight)

{

glViewport(0, 0, newWidth, newHeight); glMatrixMode(GL\_PROJECTION); glLoadIdentity();

gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax); glClear(GL\_COLOR\_BUFFER\_BIT);

}

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

{ glutInit(&argc, argv); glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB); glutInitWindowPosition(50, 50); glutInitWindowSize(winWidth, winHeight); glutCreateWindow("Bezier Curve"); int a\_menu=glutCreateMenu(cmenu); glutAddMenuEntry("start", 1); glutAddMenuEntry("stop", 2); glutCreateMenu(handlemenu); glutAddSubMenu("animate", a\_menu); glutAddMenuEntry("Quit",4); glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutDisplayFunc(displayFcn); glutReshapeFunc(winReshapeFun); glutMainLoop(); }

**Variant 4:**

#include"stdafx.h"

#include<GL/glut.h>

#include<stdio.h>

#include<math.h>

static int window;

static int menu\_id;

static int submenu\_id;

static int value = 0;

void menu(int num){

if (num==1)

exit(0);

else{

value = num;

}

glutPostRedisplay();

}

#define PI 3.1416

GLsizei winWidth = 600, winHeight = 600;

GLfloat xwcMin = 0.0, xwcMax = 130.0;

GLfloat ywcMin = 0.0, ywcMax = 130.0;

typedef struct wcPt3D

{

GLfloat x, y, z;

};

void createMenu(void)

{

submenu\_id = glutCreateMenu(menu);

glutAddMenuEntry("Flag", 2);

menu\_id = glutCreateMenu(menu);

glutAddSubMenu("Draw", submenu\_id);

glutAddMenuEntry("Quit", 1);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

void bino(GLint n, GLint \*C)

{

GLint k, j;

for(k=0;k<=n;k++)

{

C[k]=1;

for(j=n;j>=k+1; j--)

C[k]\*=j;

for(j=n-k;j>=2;j--)

C[k]/=j;

}

}

void computeBezPt(GLfloat u, wcPt3D \*bezPt, GLint nCtrlPts, wcPt3D \*ctrlPts, GLint \*C)

{

GLint k, n=nCtrlPts-1;

GLfloat bezBlendFcn;

bezPt ->x =bezPt ->y = bezPt->z=0.0;

for(k=0; k< nCtrlPts; k++)

{

bezBlendFcn = C[k] \* pow(u, k) \* pow( 1-u, n-k);

bezPt ->x += ctrlPts[k].x \* bezBlendFcn;

bezPt ->y += ctrlPts[k].y \* bezBlendFcn;

bezPt ->z += ctrlPts[k].z \* bezBlendFcn;

}

}

void bezier(wcPt3D \*ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)

{

wcPt3D bezCurvePt;

GLfloat u;

GLint \*C, k;

C= new GLint[nCtrlPts];

bino(nCtrlPts-1, C);

glBegin(GL\_LINE\_STRIP);

for(k=0; k<=nBezCurvePts; k++)

{

u=GLfloat(k)/GLfloat(nBezCurvePts);

computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);

glVertex2f(bezCurvePt.x, bezCurvePt.y);

}

glEnd();

delete[]C;

}

void displayFcn()

{

GLint nCtrlPts = 4, nBezCurvePts =20;

static float theta = 0;

wcPt3D ctrlPts[4] = {

{20, 100, 0},

{30, 110, 0},

{50, 90, 0},

{60, 100, 0}};

ctrlPts[1].x +=10\*sin(theta \* PI/180.0);

ctrlPts[1].y +=5\*sin(theta \* PI/180.0);

ctrlPts[2].x -= 10\*sin((theta+30) \* PI/180.0);

ctrlPts[2].y -= 10\*sin((theta+30) \* PI/180.0);

ctrlPts[3].x-= 4\*sin((theta) \* PI/180.0);

ctrlPts[3].y += sin((theta-30) \* PI/180.0);

theta+=0.1;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 1.0, 1.0);

glPointSize(5);

glPushMatrix();

glLineWidth(5);

glColor3f(255/255, 153/255.0, 51/255.0); //Indian flag: Orange color code

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

{

glTranslatef(0, -0.8, 0);

bezier(ctrlPts, nCtrlPts, nBezCurvePts);

}

glColor3f(1, 1, 1);

//Indian flag: white color code

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

{

glTranslatef(0, -0.8, 0);

bezier(ctrlPts, nCtrlPts, nBezCurvePts);

}

glColor3f(19/255.0, 136/255.0, 8/255.0); //Indian flag: green color code

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

{

glTranslatef(0, -0.8, 0);

bezier(ctrlPts, nCtrlPts, nBezCurvePts);

}

glPopMatrix();

glColor3f(0.7, 0.5,0.3);

glLineWidth(5);

glBegin(GL\_LINES);

glVertex2f(20,100);

glVertex2f(20,40);

glEnd();

glFlush();

glutPostRedisplay();

glutSwapBuffers();

}

void winReshapeFun(GLint newWidth, GLint newHeight)

{

glViewport(0, 0, newWidth, newHeight);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);

glClear(GL\_COLOR\_BUFFER\_BIT);

}

void display(void){

glClear(GL\_COLOR\_BUFFER\_BIT);

if(value == 1){

return; //glutPostRedisplay()

}else if(value == 2)

displayFcn();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowPosition(50, 50);

glutInitWindowSize(winWidth, winHeight);

glutCreateWindow("Bezier Curve");

createMenu();

glutDisplayFunc(display);

glutReshapeFunc(winReshapeFun);

glutMainLoop();

}

1. **Develop a menu driven program to fill the polygon using scan line algorithm**

**Variant 1**

#include<stdlib.h>  
#include<stdio.h>  
#include<GL/glut.h>  
float x1,x2,x3,x4,y1,y2,y3,y4;

void edgedetect(float x1,float y1,float x2,float y2,int \*le,int \*re)  
{  
float mx,x,temp;  
int i;  
if((y2-y1)<0)  
{  
temp=y1;  
y1=y2;  
y2=temp;

temp=x1;  
x1=x2;  
x2=temp;  
}

if((y2-y1)!=0)  
mx=(x2-x1)/(y2-y1);  
else  
mx=x2-x1;  
x=x1;

for(i=y1;i<=y2;i++)  
{  
if(x<(float)le[i])  
le[i]=(int)x;

if(x>(float)re[i])  
re[i]=(int)x;  
x+=mx;  
}  
}

void draw\_pixel(int x,int y)  
{  
glColor3f(1.0,1.0,0.0);  
glBegin(GL\_POINTS);  
glVertex2i(x,y);  
glEnd();  
}

void scanfill(float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)  
{  
int le[500],re[500];  
int i,y;  
for(i=0;i<500;i++)  
{  
le[i]=500;  
re[i]=0;  
}  
edgedetect(x1,y1,x2,y2,le,re);  
edgedetect(x2,y2,x3,y3,le,re);  
edgedetect(x3,y3,x4,y4,le,re);  
edgedetect(x4,y4,x1,y1,le,re);  
for(y=0;y<500;y++)  
{  
if(le[y]<=re[y])  
for(i=(int)le[y];i<(int)re[y];i++)  
draw\_pixel(i,y);  
}  
}

void display()  
{  
x1=200.0;  
y1=200.0;

x2=100.0;  
y2=300.0;

x3=200.0;  
y3=400.0;

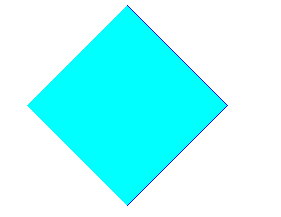
x4=300.0;  
y4=300.0;

glClear(GL\_COLOR\_BUFFER\_BIT);  
glColor3f(1.0,0.0,0.0);  
glBegin(GL\_LINE\_LOOP);  
glVertex2f(x1,y1);  
glVertex2f(x2,y2);  
glVertex2f(x3,y3);  
glVertex2f(x4,y4);  
glEnd();  
scanfill(x1,y1,x2,y2,x3,y3,x4,y4);  
glFlush();  
}

void myinit()  
{  
glClearColor(1.0,1.0,1.0,1.0);  
glColor3f(1.0,0.0,0.0);  
glPointSize(1.0);  
glMatrixMode(GL\_PROJECTION);  
glLoadIdentity();  
gluOrtho2D(0.0,499.0,0.0,499.0);  
}

int main(int argc,char\*\* argv)  
{  
glutInit(&argc,argv);  
glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  
glutInitWindowSize(500,500);  
glutInitWindowPosition(0,0);  
glutCreateWindow("Scan line area filling algorithm");  
glutDisplayFunc(display);  
myinit();  
glutMainLoop();  
return 0;  
}

**Output:**

****

**Variant 2:**

#include"stdafx.h"

#define BLACK 0

#include<stdlib.h>

#include<stdio.h>

#include<GL/glut.h>

float x1,x2,x3,x4,y1,y2,y3,y4;

static int window;

static int menu\_id;

static int submenu\_id;

static int value = 0;

void menu(int num){

if (num==1)

exit(0);

else{

value = num;

}

glutPostRedisplay();

}

void edgedetect(float x1,float y1,float x2,float y2,int \*le,int \*re)

{

float mx,x,temp;

int i;

if((y2-y1)<0)   // if second point is below first point interchange them

{

temp=y1;y1=y2;y2=temp;

temp=x1;x1=x2;x2=temp;

}

if((y2-y1)!=0)  // if denominator is zero we can't find slope

mx=(x2-x1)/(y2-y1);

else

mx=x2-x1; // y2-y1=0 implies line is horizontal

x=x1;

for(i=y1;i<=y2;i++) // starting from x1,y1 add slope mx to x  
           // and round it to find the next point on the  
                                                // line. For that particular scan line i

{

if(x<(float)le[i])  // insert the x value into either le or re.

le[i]=(int)x; // Initially both le and re will contain same value...

if(x>(float)re[i]) // in the next time for the other edge

re[i]=(int)x; // either le or re will change

x+=mx;  // NOTE: le and re are integer arrays and x  
                                       // is float so automatic type conversion.  
}

}

}

void draw\_pixel(int x,int y,int value)

{

glColor3f(1.0,1.0,0.0);

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

void scanfill(float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)

{

int le[500],re[500];

int i,y;

for(i=0;i<500;i++)  // initialize le and re array values

{

le[i]=500;

re[i]=0;

}

edgedetect(x1,y1,x2,y2,le,re); // call edge detect four times

edgedetect(x2,y2,x3,y3,le,re);   // once for each edge.

edgedetect(x3,y3,x4,y4,le,re);

edgedetect(x4,y4,x1,y1,le,re);

for(y=0;y<500;y++)  // for every scan line with value y

{

if(le[y]<=re[y])  // refer to le and re arrays to see if a part

for(i=(int)le[y];i<(int)re[y];i++) // of the scanline is inside polygon

draw\_pixel(i,y,BLACK); // if so draw a horizontal line from  
                                                           // left edge to right edge

}

}

void displayFcn()

{

x1=200.0;y1=200.0;x2=100.0;y2=300.0;x3=200.0;y3=400.0;x4=300.0;y4=300.0;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_LOOP);  // draw the boundary of the polygon to be filled.

glVertex2f(x1,y1);

glVertex2f(x2,y2);

glVertex2f(x3,y3);

glVertex2f(x4,y4);

glEnd();

scanfill(x1,y1,x2,y2,x3,y3,x4,y4); // call scanfill to fill the polygon

glFlush();// Usually students fail the lab because they forget glFlush.

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

if(value == 1){

return;

}else if(value == 2)

displayFcn();

else if (value=3)

scanfill(x1,y1,x2,y2,x3,y3,x4,y4);

}

void myinit()

{

glClearColor(1.0,1.0,1.0,1.0);

glColor3f(1.0,0.0,0.0);

glPointSize(1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,499.0,0.0,499.0);

}

void createMenu(void)

{

menu\_id = glutCreateMenu(menu);

glutAddMenuEntry("Quit", 1);

glutAddMenuEntry("draw polygon", 2);

glutAddMenuEntry("fill polygon", 3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500,500);

glutInitWindowPosition(0,0);

glutCreateWindow("Filling a Polygon using Scan-line Algorithm");

createMenu();

glutDisplayFunc(display);

myinit();

glutMainLoop();

}

**Variant 3:**

#include<windows.h>

#include<GL/glu.h>

#include<GL/glut.h>

int LE[500],RE[500];

int EdgeFlag = 0,FillFlag = 0;

void Intersection(GLint x1,GLint y1,GLint x2,GLint y2)

{

float x,M;

int t,y;

if(y1>y2)

{

t = x1;

x1 = x2;

x2 = t;

t = y1;

y1 = y2;

y2 = t;

}

if((y2-y1)==0)

M = (x2-x1);

else

M = (x2-x1)/(y2-y1);

x = x1;

for(y=y1;y<=y2;y++)

{

if(x<LE[y])

LE[y]=x;

if(x>RE[y])

RE[y]=x;

x = x + M;

}

}

void Draw()

{

int x,y,i;

GLint P1[2] = {125,250},P2[2] = {250,125},P3[2] = {375,250},P4[2] = {250,375};

glClear(GL\_COLOR\_BUFFER\_BIT);

for(i=0;i<500;i++)

{

LE[i] = 500;

RE[i] = 0;

}

if(EdgeFlag == 1)

{

glBegin(GL\_LINE\_LOOP);

glVertex2iv(P1);

glVertex2iv(P2);

glVertex2iv(P3);

glVertex2iv(P4);

glEnd();

}

Intersection(P1[0],P1[1],P2[0],P2[1]);

Intersection(P2[0],P2[1],P3[0],P3[1]);

Intersection(P3[0],P3[1],P4[0],P4[1]);

Intersection(P4[0],P4[1],P1[0],P1[1]);

if(FillFlag == 1)

{

for(y=0;y<500;y++)

{

for(x=LE[y];x<RE[y];x++)

{

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

glFlush();

}

}

}

glFlush();

}

void Menu(int id)

{

if(id == 1)

EdgeFlag = 1;

else if(id == 2)

EdgeFlag = 0;

else if(id == 3)

exit(0);

FillFlag = 1;

glutPostRedisplay();

}

void MyInit()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0,500,0,500);

glMatrixMode(GL\_MODELVIEW);

glutCreateMenu(Menu);

glutAddMenuEntry("With Edge",1);

glutAddMenuEntry("Without Edge",2);

glutAddMenuEntry("Exit",3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

int main(int argC,char \*argV[])

{

glutInit(&argC,argV);

glutInitWindowSize(500,500);

glutInitWindowPosition(100,100);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutCreateWindow("Polygon Fill");

MyInit();

glutDisplayFunc(Draw);

glutMainLoop();

return 0;

}

**Variant 4:**

// CPP program to illustrate

// Scanline Polygon fill Algorithm

#include <stdio.h>

#include <math.h>

#include <GL/glut.h>

#define maxHt 800

#define maxWd 600

#define maxVer 10000

FILE \*fp;

// Start from lower left corner

typedef struct edgebucket

{

int ymax; //max y-coordinate of edge

float xofymin; //x-coordinate of lowest edge point updated only in aet

float slopeinverse;

}EdgeBucket;

typedef struct edgetabletup

{

// the array will give the scanline number

// The edge table (ET) with edges entries sorted

// in increasing y and x of the lower end

int countEdgeBucket; //no. of edgebuckets

EdgeBucket buckets[maxVer];

}EdgeTableTuple;

EdgeTableTuple EdgeTable[maxHt], ActiveEdgeTuple;

// Scanline Function

void initEdgeTable()

{

int i;

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

{

EdgeTable[i].countEdgeBucket = 0;

}

ActiveEdgeTuple.countEdgeBucket = 0;

}

void printTuple(EdgeTableTuple \*tup)

{

int j;

if (tup->countEdgeBucket)

printf("\nCount %d-----\n",tup->countEdgeBucket);

for (j=0; j<tup->countEdgeBucket; j++)

{

printf(" %d+%.2f+%.2f",

tup->buckets[j].ymax, tup->buckets[j].xofymin,tup->buckets[j].slopeinverse);

}

}

void printTable()

{

int i,j;

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

{

if (EdgeTable[i].countEdgeBucket)

printf("\nScanline %d", i);

printTuple(&EdgeTable[i]);

}

}

/\* Function to sort an array using insertion sort\*/

void insertionSort(EdgeTableTuple \*ett)

{

int i,j;

EdgeBucket temp;

for (i = 1; i < ett->countEdgeBucket; i++)

{

temp.ymax = ett->buckets[i].ymax;

temp.xofymin = ett->buckets[i].xofymin;

temp.slopeinverse = ett->buckets[i].slopeinverse;

j = i - 1;

while ((temp.xofymin < ett->buckets[j].xofymin) && (j >= 0))

{

ett->buckets[j + 1].ymax = ett->buckets[j].ymax;

ett->buckets[j + 1].xofymin = ett->buckets[j].xofymin;

ett->buckets[j + 1].slopeinverse = ett->buckets[j].slopeinverse;

j = j - 1;

}

ett->buckets[j + 1].ymax = temp.ymax;

ett->buckets[j + 1].xofymin = temp.xofymin;

ett->buckets[j + 1].slopeinverse = temp.slopeinverse;

}

}

void storeEdgeInTuple (EdgeTableTuple \*receiver,int ym,int xm,float slopInv)

{

// both used for edgetable and active edge table..

// The edge tuple sorted in increasing ymax and x of the lower end.

(receiver->buckets[(receiver)->countEdgeBucket]).ymax = ym;

(receiver->buckets[(receiver)->countEdgeBucket]).xofymin = (float)xm;

(receiver->buckets[(receiver)->countEdgeBucket]).slopeinverse = slopInv;

// sort the buckets

insertionSort(receiver);

(receiver->countEdgeBucket)++;

}

void storeEdgeInTable (int x1,int y1, int x2, int y2)

{

float m,minv;

int ymaxTS,xwithyminTS, scanline; //ts stands for to store

if (x2==x1)

{

minv=0.000000;

}

else

{

m = ((float)(y2-y1))/((float)(x2-x1));

// horizontal lines are not stored in edge table

if (y2==y1)

return;

minv = (float)1.0/m;

printf("\nSlope string for %d %d & %d %d: %f",x1,y1,x2,y2,minv);

}

if (y1>y2)

{

scanline=y2;

ymaxTS=y1;

xwithyminTS=x2;

}

else

{

scanline=y1;

ymaxTS=y2;

xwithyminTS=x1;

}

// the assignment part is done..now storage..

storeEdgeInTuple(&EdgeTable[scanline],ymaxTS,xwithyminTS,minv);

}

void removeEdgeByYmax(EdgeTableTuple \*Tup,int yy)

{

int i,j;

for (i=0; i< Tup->countEdgeBucket; i++)

{

if (Tup->buckets[i].ymax == yy)

{

printf("\nRemoved at %d",yy);

for ( j = i ; j < Tup->countEdgeBucket -1 ; j++ )

{

Tup->buckets[j].ymax =Tup->buckets[j+1].ymax;

Tup->buckets[j].xofymin =Tup->buckets[j+1].xofymin;

Tup->buckets[j].slopeinverse = Tup->buckets[j+1].slopeinverse;

}

Tup->countEdgeBucket--;

i--;

}

}

}

void updatexbyslopeinv(EdgeTableTuple \*Tup)

{

int i;

for (i=0; i<Tup->countEdgeBucket; i++)

{

(Tup->buckets[i]).xofymin =(Tup->buckets[i]).xofymin + (Tup->buckets[i]).slopeinverse;

}

}

void ScanlineFill()

{

/\* Follow the following rules:

1. Horizontal edges: Do not include in edge table

2. Horizontal edges: Drawn either on the bottom or on the top.

3. Vertices: If local max or min, then count twice, else count

once.

4. Either vertices at local minima or at local maxima are drawn.\*/

int i, j, x1, ymax1, x2, ymax2, FillFlag = 0, coordCount;

// we will start from scanline 0;

// Repeat until last scanline:

for (i=0; i<maxHt; i++)//4. Increment y by 1 (next scan line)

{

// 1. Move from ET bucket y to the

// AET those edges whose ymin = y (entering edges)

for (j=0; j<EdgeTable[i].countEdgeBucket; j++)

{

storeEdgeInTuple(&ActiveEdgeTuple,EdgeTable[i].buckets[j].

ymax,EdgeTable[i].buckets[j].xofymin,

EdgeTable[i].buckets[j].slopeinverse);

}

printTuple(&ActiveEdgeTuple);

// 2. Remove from AET those edges for

// which y=ymax (not involved in next scan line)

removeEdgeByYmax(&ActiveEdgeTuple, i);

//sort AET (remember: ET is presorted)

insertionSort(&ActiveEdgeTuple);

printTuple(&ActiveEdgeTuple);

//3. Fill lines on scan line y by using pairs of x-coords from AET

j = 0;

FillFlag = 0;

coordCount = 0;

x1 = 0;

x2 = 0;

ymax1 = 0;

ymax2 = 0;

while (j<ActiveEdgeTuple.countEdgeBucket)

{

if (coordCount%2==0)

{

x1 = (int)(ActiveEdgeTuple.buckets[j].xofymin);

ymax1 = ActiveEdgeTuple.buckets[j].ymax;

if (x1==x2)

{

/\* three cases can arrive-

1. lines are towards top of the intersection

2. lines are towards bottom

3. one line is towards top and other is towards bottom

\*/

if (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)))

{

x2 = x1;

ymax2 = ymax1;

}

else

{

coordCount++;

}

}

else

{

coordCount++;

}

}

else

{

x2 = (int)ActiveEdgeTuple.buckets[j].xofymin;

ymax2 = ActiveEdgeTuple.buckets[j].ymax;

FillFlag = 0;

// checking for intersection...

if (x1==x2)

{

/\*three cases can arrive-

1. lines are towards top of the intersection

2. lines are towards bottom

3. one line is towards top and other is towards bottom

\*/

if (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)))

{

x1 = x2;

ymax1 = ymax2;

}

else

{

coordCount++;

FillFlag = 1;

}

}

else

{

coordCount++;

FillFlag = 1;

}

if(FillFlag)

{

//drawing actual lines...

glColor3f(0.0f,0.7f,0.0f);

glBegin(GL\_LINES);

glVertex2i(x1,i);

glVertex2i(x2,i);

glEnd();

glFlush();

// printf("\nLine drawn from %d,%d to %d,%d",x1,i,x2,i);

}

}

j++;

}

// 5. For each nonvertical edge remaining in AET, update x for new y

updatexbyslopeinv(&ActiveEdgeTuple);

}

printf("\nScanline filling complete");

}

void myInit(void)

{

glClearColor(1.0,1.0,1.0,0.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0,maxHt,0,maxWd);

glClear(GL\_COLOR\_BUFFER\_BIT);

}

void drawPolyDino()

{

glColor3f(1.0f,0.0f,0.0f);

int count = 0,x1,y1,x2,y2;

rewind(fp);

while(!feof(fp) )

{

count++;

if (count>2)

{

x1 = x2;

y1 = y2;

count=2;

}

if (count==1)

{

fscanf(fp, "%d,%d", &x1, &y1);

}

else

{

fscanf(fp, "%d,%d", &x2, &y2);

printf("\n%d,%d", x2, y2);

glBegin(GL\_LINES);

glVertex2i( x1, y1);

glVertex2i( x2, y2);

glEnd();

storeEdgeInTable(x1, y1, x2, y2);//storage of edges in edge table.

glFlush();

}

}

}

void drawDino(void)

{

initEdgeTable();

drawPolyDino();

printf("\nTable");

printTable();

ScanlineFill();//actual calling of scanline filling..

}

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

{

fp=fopen ("PolyDino.txt","r");

if ( fp == NULL )

{

printf( "Could not open file" ) ;

return;

}

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(maxHt,maxWd);

glutInitWindowPosition(100, 150);

glutCreateWindow("Scanline filled dinosaur");

myInit();

glutDisplayFunc(drawDino);

glutMainLoop();

fclose(fp);

}

**VIVA QUESTIONS:**

* What is scan conversion?
* Write the properties of video display devices?
* What is rasterization?
* Define Computer graphics.
* Name any four input devices
* Write the two techniques for producing color displayswith a CRT?
* What is vertical retrace of the electron beam?
* Short notes on video controller?
* What is bitmap?
* Differentiate plasma panel display and thin film electroluminescent display?
* What is resolution?
* What is horizontal retrace of the electron beam?
* What is filament?
* What is pixmap?
* Write the types of clipping?
* What is meant by scan code?
* List out the merits and demerits of Penetrationtechniques?
* List out the merits and demerits of DVST?
* What do you mean by emissive and non-emissivedisplays?
* List out the merits and demerits of Plasma paneldisplay?
* What is persistence?
* What is Aspect ratio?
* What is the difference between impact and non-impactprinters?
* Define pixel?
* What is frame buffer?.
* Where the video controller is used?
* What is run length encoding?.
* What is point in the computer graphics system?
* Write short notes on lines?
* Define Circle?
* What are the various attributes of a line?
* What is antialiasing?.
* What is Transformation?
* What is translation?
* What is rotation?
* What is scaling?
* What is shearing?.
* What is reflection?
* What are the two classifications of sheartransformation?
* A point (4,3) is rotated counterclockwise by an angle of45°. Find the rotation matrix and the resultant point