**Name: Durva Rajesh Ajgaonkar**

**Class: CS-A**

**Roll no: 8**

**PRN no: 12210125**

**Line Clipping Algorithms**

1. **Liang-Barsky line clipping algorithm**

#include <GL/glut.h>

#include <stdio.h>

void liangBarsky(float x0, float y0, float x1, float y1, float xmin, float ymin, float xmax, float ymax) {

float t0 = 0.0, t1 = 1.0;

float dx = x1 - x0, dy = y1 - y0;

float p, q, r;

for (int i = 0; i < 4; ++i) {

switch (i) {

case 0: p = -dx; q = -(xmin - x0); break;

case 1: p = dx; q = (xmax - x0); break;

case 2: p = -dy; q = -(ymin - y0); break;

case 3: p = dy; q = (ymax - y0); break;

}

r = q / p;

if (p == 0 && q < 0) {

printf("Line is parallel and outside the boundary\n");

return;

}

if (p < 0) {

if (r > t1)

printf("Line is outside the boundary\n");

else if (r > t0)

t0 = r;

} else if (p > 0) {

if (r < t0)

printf("Line is outside the boundary\n");

else if (r < t1)

t1 = r;

}

}

if (t0 < t1) {

float clipped\_x0 = x0 + t0 \* dx;

float clipped\_y0 = y0 + t0 \* dy;

float clipped\_x1 = x0 + t1 \* dx;

float clipped\_y1 = y0 + t1 \* dy;

glColor3f(1.0, 0.0, 0.0); // Red color for clipped line

glBegin(GL\_LINES);

glVertex2f(clipped\_x0, clipped\_y0);

glVertex2f(clipped\_x1, clipped\_y1);

glEnd();

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 1.0, 0.0); // Green color for original line

glBegin(GL\_LINES);

glVertex2f(10, 10);

glVertex2f(100, 100);

glEnd();

liangBarsky(10, 10, 100, 100, 20, 20, 80, 80); // Clipping window (20, 20) to (80, 80)

glFlush();

}

void init() {

glClearColor(0.0, 0.0, 0.0, 1.0);

gluOrtho2D(0, 500, 0, 500);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Liang-Barsky Line Clipping");

init();

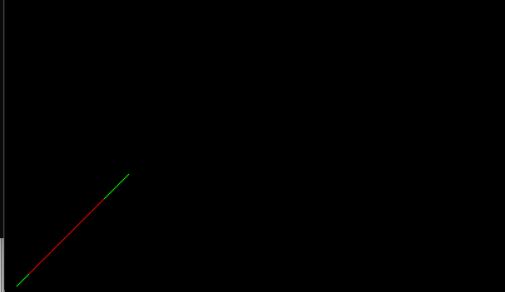
glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**Output:**

****

1. **Cohen Sutherland Algorithm**

#include <windows.h>

#include<GL/glut.h>

#include<math.h>

#include<stdio.h>

#include<iostream>

void display();

using namespace std;

float xmin=-100;

float ymin=-100;

float xmax=100;

float ymax=100;

float xd1,yd1,xd2,yd2;

void init(void)

{

glClearColor(0.0,0,0,0);

glMatrixMode(GL\_PROJECTION);

glOrtho(0.0, 640.0, 0.0, 640.0, -1.0, 1.0);

}

int code(float x,float y)

{

int c=0;

if(y>ymax)c=8;

if(y<ymin)c=4;

if(x>xmax)c=c|2;

if(x<xmin)c=c|1;

return c;

}

void cohen\_Line(float x1,float y1,float x2,float y2)

{

int c1=code(x1,y1);

int c2=code(x2,y2);

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

while((c1|c2)>0)

{

if((c1 & c2)>0)

{

exit(0);

}

float xi=x1;float yi=y1;

int c=c1;

if(c==0)

{

c=c2;

xi=x2;

yi=y2;

}

float x,y;

if((c & 8)>0)

{

y=ymax;

x=xi+ 1.0/m\*(ymax-yi);

}

else

if((c & 4)>0)

{

y=ymin;

x=xi+1.0/m\*(ymin-yi);

}

else

if((c & 2)>0)

{

x=xmax;

y=yi+m\*(xmax-xi);

}

else

if((c & 1)>0)

{

x=xmin;

y=yi+m\*(xmin-xi);

}

if(c==c1)

{

xd1=x;

yd1=y;

c1=code(xd1,yd1);

}

if(c==c2)

{

xd2=x;

yd2=y;

c2=code(xd2,yd2);

}

}

display();

}

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

{

if(key=='c')

{ cout<<"Hello";

cohen\_Line(xd1,yd1,xd2,yd2);

glFlush();

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0,1.0,0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2i(xmin,ymin);

glVertex2i(xmin,ymax);

glVertex2i(xmax,ymax);

glVertex2i(xmax,ymin);

glEnd();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex2i(xd1,yd1);

glVertex2i(xd2,yd2);

glEnd();

glFlush();

}

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

{

printf("Enter line co-ordinates:");

cin>>xd1>>yd1>>xd2>>yd2;

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(600,600);

glutInitWindowPosition(0,0);

glutCreateWindow("Clipping");

glutDisplayFunc(display);

glutKeyboardFunc(mykey);

init();

glutMainLoop();

return 0;

}

**Output:**

1. **Cyrus Beck Algorithm:**

#include <windows.h>

#include<GL/glut.h>

#include<math.h>

#include<stdio.h>

#include<iostream>

void display();

using namespace std;

float xmin=-100;

float ymin=-100;

float xmax=100;

float ymax=100;

float xd1,yd1,xd2,yd2;

void init(void)

{

glClearColor(0.0,0,0,0);

glMatrixMode(GL\_PROJECTION);

glOrtho(0.0, 640.0, 0.0, 640.0, -1.0, 1.0);

}

int code(float x,float y)

{

int c=0;

if(y>ymax)c=8;

if(y<ymin)c=4;

if(x>xmax)c=c|2;

if(x<xmin)c=c|1;

return c;

}

void cohen\_Line(float x1,float y1,float x2,float y2)

{

int c1=code(x1,y1);

int c2=code(x2,y2);

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

while((c1|c2)>0)

{

if((c1 & c2)>0)

{

exit(0);

}

float xi=x1;float yi=y1;

int c=c1;

if(c==0)

{

c=c2;

xi=x2;

yi=y2;

}

float x,y;

if((c & 8)>0)

{

y=ymax;

x=xi+ 1.0/m\*(ymax-yi);

}

else

if((c & 4)>0)

{

y=ymin;

x=xi+1.0/m\*(ymin-yi);

}

else

if((c & 2)>0)

{

x=xmax;

y=yi+m\*(xmax-xi);

}

else

if((c & 1)>0)

{

x=xmin;

y=yi+m\*(xmin-xi);

}

if(c==c1)

{

xd1=x;

yd1=y;

c1=code(xd1,yd1);

}

if(c==c2)

{

xd2=x;

yd2=y;

c2=code(xd2,yd2);

}

}

display();

}

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

{

if(key=='c')

{ cout<<"Hello";

cohen\_Line(xd1,yd1,xd2,yd2);

glFlush();

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0,1.0,0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2i(xmin,ymin);

glVertex2i(xmin,ymax);

glVertex2i(xmax,ymax);

glVertex2i(xmax,ymin);

glEnd();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex2i(xd1,yd1);

glVertex2i(xd2,yd2);

glEnd();

glFlush();

}

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

{

printf("Enter line co-ordinates:");

cin>>xd1>>yd1>>xd2>>yd2;

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(600,600);

glutInitWindowPosition(0,0);

glutCreateWindow("Clipping");

glutDisplayFunc(display);

glutKeyboardFunc(mykey);

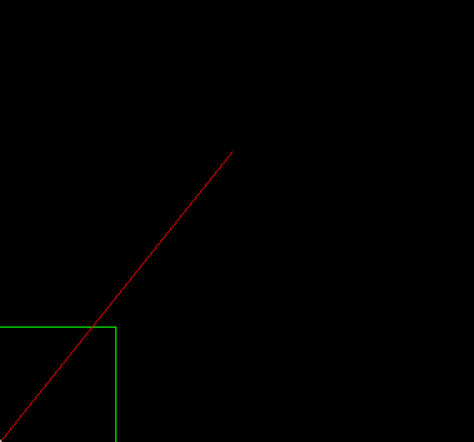
init();

glutMainLoop();

return 0;

}

**Output:**



**2D Transformations**

#include <GL/glut.h>

#include <stdbool.h>

#include <unistd.h>

float angle = 0.0f;

float shearX = 0.0f;

float shearY = 0.0f;

float scaleX = 1.0f;

float scaleY = 1.0f;

float translateX = 0.0f;

float translateY = 0.0f;

bool rotationDone = false;

bool reflectionDone = false;

bool shearingDone = false;

bool scalingDone = false;

bool translationDone = false;

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glTranslatef(translateX, translateY, 0.0f);

glRotatef(angle, 0.0f, 0.0f, 1.0f);

glScalef(scaleX, scaleY, 1.0f);

glTranslatef(translateX \* shearX, translateY \* shearY, 0.0f);

if (reflectionDone) {

glScalef(-1.0f, 1.0f, 1.0f);

}

glBegin(GL\_QUADS);

glColor3f(1.0f, 0.0f, 0.0f);

glVertex2f(-0.5f, -0.5f);

glVertex2f(0.5f, -0.5f);

glVertex2f(0.5f, 0.5f);

glVertex2f(-0.5f, 0.5f);

glEnd();

glutSwapBuffers();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-1.0, 1.0, -1.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

}

void animate(int value) {

if (!rotationDone) {

angle += 1.0f;

if (angle >= 90.0f) {

angle = 90.0f;

rotationDone = true;

usleep(1000000); // 1 second delay after rotation

}

} else if (!reflectionDone) {

glScalef(-1.0f, 1.0f, 1.0f);

reflectionDone = true;

} else if (!shearingDone) {

shearX += 0.05f;

shearY += 0.05f;

if (shearX >= 0.2f) {

shearX = 0.2f;

shearY = 0.2f;

shearingDone = true;

usleep(2000000);

}

} else if (!scalingDone) {

scaleX = 0.5f;

scaleY = 0.5f;

scalingDone = true;

usleep(2000000); // 1 second delay after scaling

} else if (!translationDone) {

translateX = 0.5f;

translateY = 0.5f;

translationDone = true;

usleep(2000000);

}

glutPostRedisplay();

glutTimerFunc(50, animate, 0);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("2D Transformations");

glutDisplayFunc(display);

glutReshapeFunc(reshape);

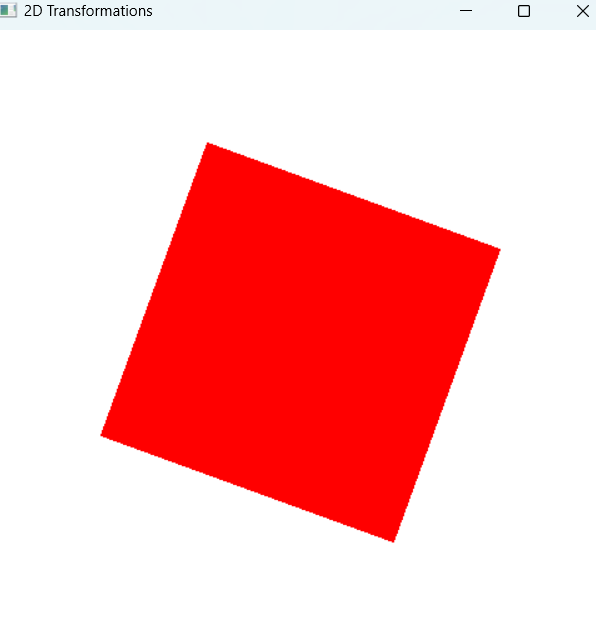
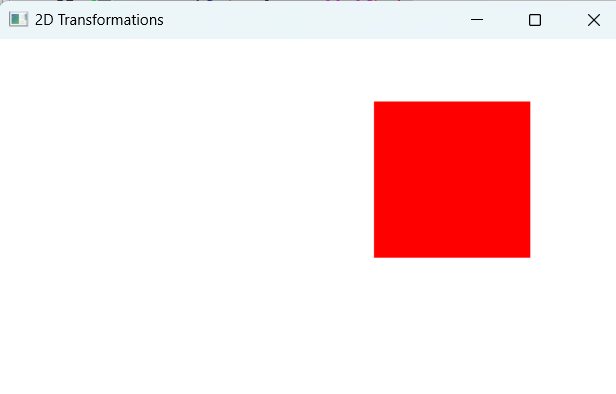
glutTimerFunc(0, animate, 0);

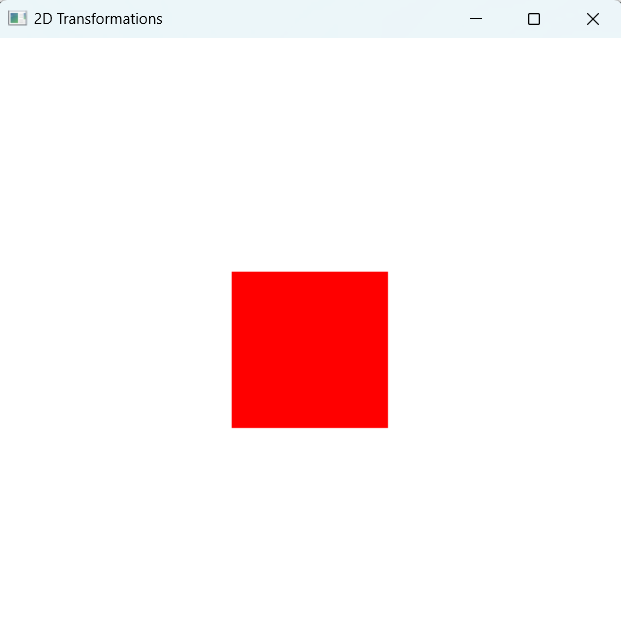
glClearColor(1.0, 1.0, 1.0, 1.0);

glutMainLoop();

return 0;

}





**2D reflection**

#include <GL/glut.h>

#include <stdio.h>

float triangleVertices[][2] = {

{0, 100}, // top

{-100, -100}, // bottom-left

{100, -100} // bottom-right

};

float reflectionVertices[][2] = {

{0, -100}, // top

{-100, 100}, // bottom-left

{100, 100} // bottom-right

};

float transitionFactor = 0.0;

void drawTriangle() {

glColor3f(1.0 - transitionFactor, 0.0, 0.0);

glBegin(GL\_TRIANGLES);

for (int i = 0; i < 3; ++i) {

glVertex2fv(triangleVertices[i]);

}

glEnd();

}

void drawReflection() {

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_TRIANGLES);

for (int i = 0; i < 3; ++i) {

float x = (1.0 - transitionFactor) \* triangleVertices[i][0] + transitionFactor \* reflectionVertices[i][0];

float y = (1.0 - transitionFactor) \* triangleVertices[i][1] + transitionFactor \* reflectionVertices[i][1];

glVertex2f(x, y);

}

glEnd();

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawTriangle();

drawReflection();

glFlush();

}

void update(int value) {

transitionFactor += 0.01;

if (transitionFactor > 1.0) {

transitionFactor = 1.0;

}

glutPostRedisplay();

glutTimerFunc(16, update, 0);

}

void init() {

glClearColor(1.0, 1.0, 1.0, 1.0); // White background

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-200, 200, -200, 200); // Set the viewing area

glMatrixMode(GL\_MODELVIEW);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(400, 400);

glutCreateWindow("2D Triangle Reflection Animation");

init();

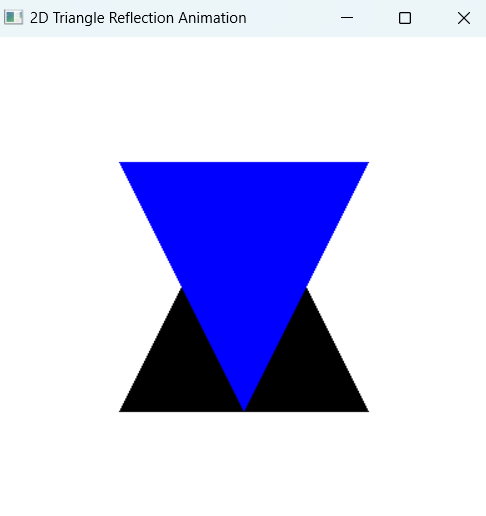
glutDisplayFunc(display);

glutTimerFunc(0, update, 0);

glutMainLoop();

return 0;

}



**3D transformations**

#include<windows.h>

#include <math.h>

#include <GL/glut.h>

#include <stdio.h>

#include <stdlib.h>

typedef float Matrix4x4 [4][4];

Matrix4x4 theMatrix;

float ptsIni[8][3]={{80,80,-100},{180,80,-100},{180,180,-100},{80,180,-100},

{60,60,0},{160,60,0},{160,160,0},{60,160,0}};

float ptsFin[8][3];

float refptX,refptY,refptZ;

float TransDistX,TransDistY,TransDistZ; //Translations along Axes

float ScaleX,ScaleY,ScaleZ; //Scaling Factors along Axes

float Alpha,Beta,Gamma,Theta; //Rotation angles about Axes

float A,B,C;

float aa,bb,cc

float x1,y11,z1,x2,y2,z2;

int choice,choiceRot,choiceRef;

void matrixSetIdentity(Matrix4x4 m)

{

int i, j;

for (i=0; i<4; i++)

for (j=0; j<4; j++)

m[i][j] = (i == j);

}

void matrixPreMultiply(Matrix4x4 a, Matrix4x4 b)

{

Matrix4x4 result;

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

result[i][j] = a[i][0] \* b[0][j] + a[i][1] \* b[1][j] + a[i][2] \* b[2][j] + a[i][3] \* b[3][j];

}

}

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

theMatrix[i][j] = result[i][j];

}

}

}

void Translate(int tx, int ty, int tz)

{

Matrix4x4 m;

matrixSetIdentity(m);

m[0][3] = tx;

m[1][3] = ty;

m[2][3] = tz;

matrixPreMultiply(m, theMatrix);

}

void Scale(float sx , float sy ,float sz)

{

Matrix4x4 m;

matrixSetIdentity(m);

m[0][0] = sx;

m[0][3] = (1 - sx)\*refptX;

m[1][1] = sy;

m[1][3] = (1 - sy)\*refptY;

m[2][2] = sz;

m[2][3] = (1 - sy)\*refptZ;

matrixPreMultiply(m , theMatrix);

}

void RotateX(float angle)

{

Matrix4x4 m;

matrixSetIdentity(m);

angle = angle\*22/1260;

m[1][1] = cos(angle);

m[1][2] = -sin(angle);

m[2][1] = sin(angle);

m[2][2] = cos(angle);

matrixPreMultiply(m , theMatrix);

}

void RotateY(float angle)

{

Matrix4x4 m;

matrixSetIdentity(m);

angle = angle\*22/1260;

m[0][0] = cos(angle);

m[0][2] = sin(angle);

m[2][0] = -sin(angle);

m[2][2] = cos(angle);

matrixPreMultiply(m , theMatrix);

}

void RotateZ(float angle)

{

Matrix4x4 m;

matrixSetIdentity(m);

angle = angle\*22/1260;

m[0][0] = cos(angle);

m[0][1] = -sin(angle);

m[1][0] = sin(angle);

m[1][1] = cos(angle);

matrixPreMultiply(m , theMatrix);

}

void Reflect(void)

{

Matrix4x4 m;

matrixSetIdentity(m);

switch(choiceRef)

{

case 1: m[2][2] = -1;

break;

case 2: m[0][0] = -1;

break;

case 3: m[1][1] = -1;

break;

}

matrixPreMultiply(m , theMatrix);

}

void DrawRotLine(void)

{

switch(choiceRot)

{

case 1: glBegin(GL\_LINES);

glVertex3s(-1000 ,B,C);

glVertex3s( 1000 ,B,C);

glEnd();

break;

case 2: glBegin(GL\_LINES);

glVertex3s(A ,-1000 ,C);

glVertex3s(A ,1000 ,C);

glEnd();

break;

case 3: glBegin(GL\_LINES);

glVertex3s(A ,B ,-1000);

glVertex3s(A ,B ,1000);

glEnd();

break;

case 4: glBegin(GL\_LINES);

glVertex3s(x1-aa\*500 ,y11-bb\*500 , z1-cc\*500);

glVertex3s(x2+aa\*500 ,y2+bb\*500 , z2+cc\*500);

glEnd();

break;

}

}

void TransformPoints(void)

{

int i,k;

float tmp ;

for(k=0 ; k<8 ; k++)

for (i=0 ; i<3 ; i++)

ptsFin[k][i] = theMatrix[i][0]\*ptsIni[k][0] + theMatrix[i][1]\*ptsIni[k][1]

+ theMatrix[i][2]\*ptsIni[k][2] + theMatrix[i][3];

}

void Axes(void)

{

glColor3f (0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2s(-1000 ,0);

glVertex2s( 1000 ,0);

glEnd();

glBegin(GL\_LINES); // Plotting Y-Axis

glVertex2s(0 ,-1000);

glVertex2s(0 , 1000);

glEnd();

}

void Draw(float a[8][3]) //Display the Figure

{

int i;

glColor3f (0.7, 0.4, 0.7);

glBegin(GL\_POLYGON);

glVertex3f(a[0][0],a[0][1],a[0][2]);

glVertex3f(a[1][0],a[1][1],a[1][2]);

glVertex3f(a[2][0],a[2][1],a[2][2]);

glVertex3f(a[3][0],a[3][1],a[3][2]);

glEnd();

i=0;

glColor3f (0.8, 0.6, 0.5);

glBegin(GL\_POLYGON);

glVertex3s(a[0+i][0],a[0+i][1],a[0+i][2]);

glVertex3s(a[1+i][0],a[1+i][1],a[1+i][2]);

glVertex3s(a[5+i][0],a[5+i][1],a[5+i][2]);

glVertex3s(a[4+i][0],a[4+i][1],a[4+i][2]);

glEnd();

glColor3f (0.2, 0.4, 0.7);

glBegin(GL\_POLYGON);

glVertex3f(a[0][0],a[0][1],a[0][2]);

glVertex3f(a[3][0],a[3][1],a[3][2]);

glVertex3f(a[7][0],a[7][1],a[7][2]);

glVertex3f(a[4][0],a[4][1],a[4][2]);

glEnd();

i=1;

glColor3f (0.5, 0.4, 0.3);

glBegin(GL\_POLYGON);

glVertex3s(a[0+i][0],a[0+i][1],a[0+i][2]);

glVertex3s(a[1+i][0],a[1+i][1],a[1+i][2]);

glVertex3s(a[5+i][0],a[5+i][1],a[5+i][2]);

glVertex3s(a[4+i][0],a[4+i][1],a[4+i][2]);

glEnd();

i=2;

glColor3f (0.5, 0.6, 0.2);

glBegin(GL\_POLYGON);

glVertex3s(a[0+i][0],a[0+i][1],a[0+i][2]);

glVertex3s(a[1+i][0],a[1+i][1],a[1+i][2]);

glVertex3s(a[5+i][0],a[5+i][1],a[5+i][2]);

glVertex3s(a[4+i][0],a[4+i][1],a[4+i][2]);

glEnd();

i=4;

glColor3f (0.7, 0.3, 0.4);

glBegin(GL\_POLYGON);

glVertex3f(a[0+i][0],a[0+i][1],a[0+i][2]);

glVertex3f(a[1+i][0],a[1+i][1],a[1+i][2]);

glVertex3f(a[2+i][0],a[2+i][1],a[2+i][2]);

glVertex3f(a[3+i][0],a[3+i][1],a[3+i][2]);

glEnd();

}

void display(void)

{

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

Axes();

glColor3f (1.0, 0.0, 0.0);

Draw(ptsIni);

matrixSetIdentity(theMatrix);

switch(choice)

{

case 1: Translate(TransDistX , TransDistY ,TransDistZ);

break;

case 2: Scale(ScaleX, ScaleY, ScaleZ);

break;

case 3: switch(choiceRot)

{

case 1: DrawRotLine();

Translate(0,-B,-C);

RotateX(Alpha);

Translate(0,B,C);

break;

case 2: DrawRotLine();

Translate(-A,0,-C);

RotateY(Beta);

Translate(A,0,C);

break;

case 3: DrawRotLine();

Translate(-A,-B,0);

RotateZ(Gamma);

Translate(A,B,0);

break;

case 4: DrawRotLine();

float MOD =sqrt((x2-x1)(x2-x1) + (y2-y11)(y2-y11) + (z2-z1)\*(z2-z1));

aa = (x2-x1)/MOD;

bb = (y2-y11)/MOD;

cc = (z2-z1)/MOD;

Translate(-x1,-y11,-z1);

float ThetaDash;

ThetaDash = 1260\*atan(bb/cc)/22;

RotateX(ThetaDash);

RotateY(1260\*asin(-aa)/22);

RotateZ(Theta);

RotateY(1260\*asin(aa)/22);

RotateX(-ThetaDash);

Translate(x1,y11,z1);

break;

}

break;

case 4: Reflect();

break;

}

TransformPoints();

Draw(ptsFin);

glFlush();

}

void init(void)

{

glClearColor (1.0, 1.0, 1.0, 1.0);

glOrtho(-454.0, 454.0, -250.0, 250.0, -250.0, 250.0);

glEnable(GL\_DEPTH\_TEST);

}

int main (int argc, char \*argv)

{

glutInit(&argc, &argv);

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

glutInitWindowSize (1362, 750);

glutInitWindowPosition (0, 0);

glutCreateWindow (" Basic Transformations ");

init ();

printf("Enter your choice number:\n1.Translation\n2.Scaling\n3.Rotation\n4.Reflection\n=>");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("Enter Translation along X, Y & Z\n=>");

scanf("%f%f%f",&TransDistX , &TransDistY , &TransDistZ);

break;

case 2:printf("Enter Scaling ratios along X, Y & Z\n=>");

scanf("%f%f%f",&ScaleX , &ScaleY , &ScaleZ);

break;

case 3:printf("Enter your choice for Rotation about axis:\n1.parallel to X-axis.(y=B & z=C)\n2.parallel to Y-axis.(x=A & z=C)\n3.parallel to Z-axis.(x=A & y=B)\n4.Arbitrary line passing through (x1,y1,z1) &(x2,y2,z2)\n =>");

scanf("%d",&choiceRot);

switch(choiceRot)

{

case 1: printf("Enter B & C: ");

scanf("%f %f",&B,&C);

printf("Enter Rot. Angle Alpha: ");

scanf("%f",&Alpha);

break;

case 2: printf("Enter A & C: ");

scanf("%f %f",&A,&C);

printf("Enter Rot. Angle Beta: ");

scanf("%f",&Beta);

break;

case 3: printf("Enter A & B: ");

scanf("%f %f",&A,&B);

printf("Enter Rot. Angle Gamma: ");

scanf("%f",&Gamma);

break;

case 4: printf("Enter values of x1 ,y1 & z1:\n");

scanf("%f %f %f",&x1,&y11,&z1);

printf("Enter values of x2 ,y2 & z2:\n");

scanf("%f %f %f",&x2,&y2,&z2);

printf("Enter Rot. Angle Theta: ");

scanf("%f",&Theta);

break;

}

break;

case 4: printf("Enter your choice for reflection about plane:\n1.X-Y\n2.Y-Z\n3.X-Z\n=>");

scanf("%d",&choiceRef);

break;

default: printf("Please enter a valid choice!!!\n");

return 0;

}

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**Output:**

