

**Computer Graphics Lab**

**Name:** Pratham Kandari

**Sap Id:** 500097663

**Batch:** 7

**Experiment -9**

**Q1) WAP to perform the 2D and 3D Transformation: Rotation ,Scaling ,Shearing ,Translation ,Reflection.**

**3D**

**CODE)**

#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}};

//Realign above line while execution

// Initial Co-ordinates ofthe Cube to be Transformed

float ptsFin[8][3];

float refptX,refptY,refptZ; //Reference points

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; //Arbitrary Line Attributes

float aa,bb,cc; //Arbitrary Line Attributes

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

int choice,choiceRot,choiceRef;

void matrixSetIdentity(Matrix4x4 m) // Initialises the matrix as Unit Matrix

{

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)

{// Multiplies matrix a times b, putting result in b

int i,j;

Matrix4x4 tmp;

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

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

tmp[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 (i = 0; i < 4; i++)

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

theMatrix[i][j] = tmp[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];

// Realign above line while execution

}

void Axes(void)

{

glColor3f (0.0, 0.0, 0.0); // Set the color to BLACK

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

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); // Set the color to RED

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);

// Set the Background color to WHITE

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

// Set the no. of Co-ordinates along X & Y axes and their gappings

glEnable(GL\_DEPTH\_TEST);

// To Render the surfaces Properly according to their depths

}

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 =>");

//Realign above line while execution

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;

}

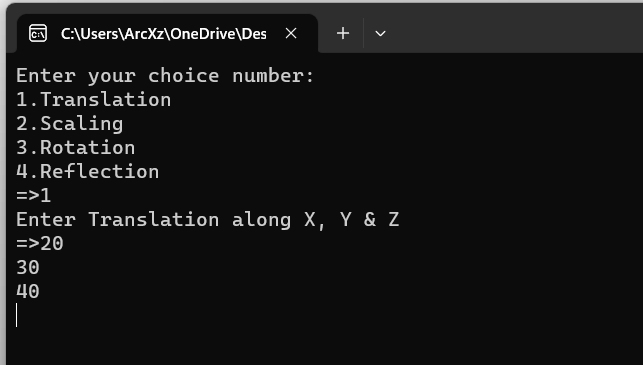
glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**3D TRANSLATION**



A picture containing shape

Description automatically generated

**3D SCALING**

Text

Description automatically generated

A picture containing graphical user interface

Description automatically generated

**3D ROTATION**

Text

Description automatically generated

A picture containing shape

Description automatically generated

**3D REFLECTION**

Text

Description automatically generated

Shape

Description automatically generated

**2D**

**CODE)**

#include <stdio.h>

#include <math.h>

#include <iostream>

#include <vector>

#include <GL/glut.h>

int choice = 0, edges;

int pntX[200],pntY[200];

int transX, transY;

double scaleX, scaleY;

double angle, angleRad;

char reflectionAxis, shearingAxis;

int shearingX, shearingY;

double round(double d)

{

return floor(d + 0.5);

}

void drawPolygon()

{

glBegin(GL\_POLYGON);

glColor3f(1.0, 0.0, 0.0);

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

{

glVertex2i(pntX[i], pntY[i]);

}

glEnd();

}

void drawPolygonTrans(int x, int y)

{

glBegin(GL\_POLYGON);

glColor3f(0.0, 1.0, 0.0);

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

{

glVertex2i(pntX[i] + x, pntY[i] + y);

}

glEnd();

}

void drawPolygonScale(double x, double y)

{

glBegin(GL\_POLYGON);

glColor3f(0.0, 0.0, 1.0);

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

{

glVertex2i(round(pntX[i] \* x), round(pntY[i] \* y));

}

glEnd();

}

void drawPolygonRotation(double angleRad)

{

glBegin(GL\_POLYGON);

glColor3f(0.0, 0.0, 1.0);

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

{

glVertex2i(round((pntX[i] \* cos(angleRad)) - (pntY[i] \* sin(angleRad))), round((pntX[i] \* sin(angleRad)) + (pntY[i] \* cos(angleRad))));

}

glEnd();

}

void drawPolygonMirrorReflection(char reflectionAxis)

{

glBegin(GL\_POLYGON);

glColor3f(0.0, 0.0, 1.0);

if (reflectionAxis == 'x' || reflectionAxis == 'X')

{

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

{

glVertex2i(round(pntX[i]), round(pntY[i] \* -1));

}

}

else if (reflectionAxis == 'y' || reflectionAxis == 'Y')

{

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

{

glVertex2i(round(pntX[i] \* -1), round(pntY[i]));

}

}

glEnd();

}

void drawPolygonShearing()

{

glBegin(GL\_POLYGON);

glColor3f(0.0, 0.0, 1.0);

if (shearingAxis == 'x' || shearingAxis == 'X')

{

glVertex2i(pntX[0], pntY[0]);

glVertex2i(pntX[1] + shearingX, pntY[1]);

glVertex2i(pntX[2] + shearingX, pntY[2]);

glVertex2i(pntX[3], pntY[3]);

}

else if (shearingAxis == 'y' || shearingAxis == 'Y')

{

glVertex2i(pntX[0], pntY[0]);

glVertex2i(pntX[1], pntY[1]);

glVertex2i(pntX[2], pntY[2] + shearingY);

glVertex2i(pntX[3], pntY[3] + shearingY);

}

glEnd();

}

void myInit(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);

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

glPointSize(4.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-640.0, 640.0, -480.0, 480.0);

}

void myDisplay(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 0.0);

if (choice == 1)

{

drawPolygon();

drawPolygonTrans(transX, transY);

}

else if (choice == 2)

{

drawPolygon();

drawPolygonScale(scaleX, scaleY);

}

else if (choice == 3)

{

drawPolygon();

drawPolygonRotation(angleRad);

}

else if (choice == 4)

{

drawPolygon();

drawPolygonMirrorReflection(reflectionAxis);

}

else if (choice == 5)

{

drawPolygon();

drawPolygonShearing();

}

glFlush();

}

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

{

printf("Enter your choice:\n\n");

printf("1. Translation \n 2. Scaling \n 3. Rotation \n 4. Mirror Reflection \n 5. Shearing \n 6. Exit" );

scanf("%d",&choice);

if (choice == 6) {

return 0;

}

printf("\n\nFor Polygon:\n");

printf( "Enter no of edges: ");

scanf("%d",&edges);

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

{

printf("Enter co-ordinates for vertex %d : ",i + 1 );

scanf("%d%d",&pntX[i],&pntY[i]);

}

if (choice == 1)

{

printf("Enter the translation factor for X and Y: ");

scanf("%d%d",&transX,&transY);

}

else if (choice == 2)

{

printf("Enter the scaling factor for X and Y: ");

scanf( "%f%f",&scaleX,&scaleY);

}

else if (choice == 3)

{

printf("Enter the angle for rotation: ");

scanf("%f",&angle);

angleRad = angle \* 3.1416 / 180;

}

else if (choice == 4)

{

printf("Enter reflection axis ( x or y ): ");

scanf("%c",&reflectionAxis);

}

else if (choice == 5)

{

printf("Enter reflection axis ( x or y ): ");

scanf("%c",&shearingAxis);

if (shearingAxis == 'x' || shearingAxis == 'X')

{

printf("Enter the shearing factor for X: ");

scanf("%d",&shearingX);

}

else

{

printf("Enter the shearing factor for Y: ");

scanf("%d",&shearingY);

}

}

//cout << "\n\nPoints:" << pntX[0] << ", " << pntY[0] << endl;

//cout << angleRad;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(640, 480);

glutInitWindowPosition(100, 150);

glutCreateWindow("Extended Basic Transformations");

glutDisplayFunc(myDisplay);

myInit();

glutMainLoop();

return 0;

}

**2D TRANSLATION**

**Text

Description automatically generated**

**Chart, waterfall chart

Description automatically generated**

**2D SCALING**

**Text

Description automatically generated**

**Chart, waterfall chart

Description automatically generated**

**2D ROTATION**

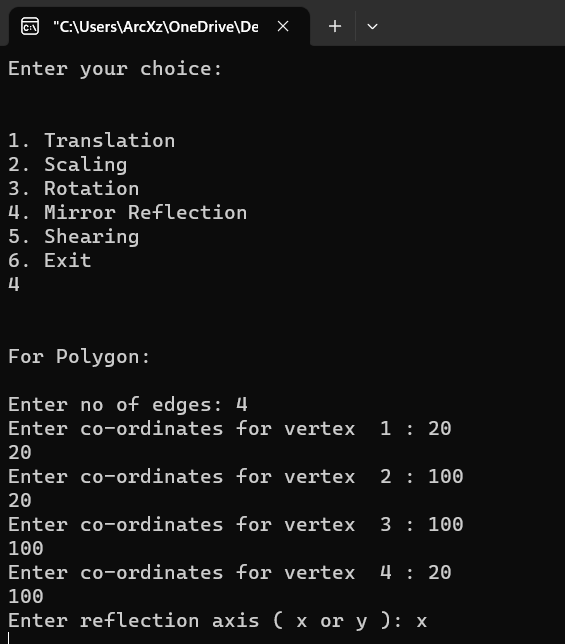
**Text

Description automatically generated**

**Graphical user interface

Description automatically generated with medium confidence**

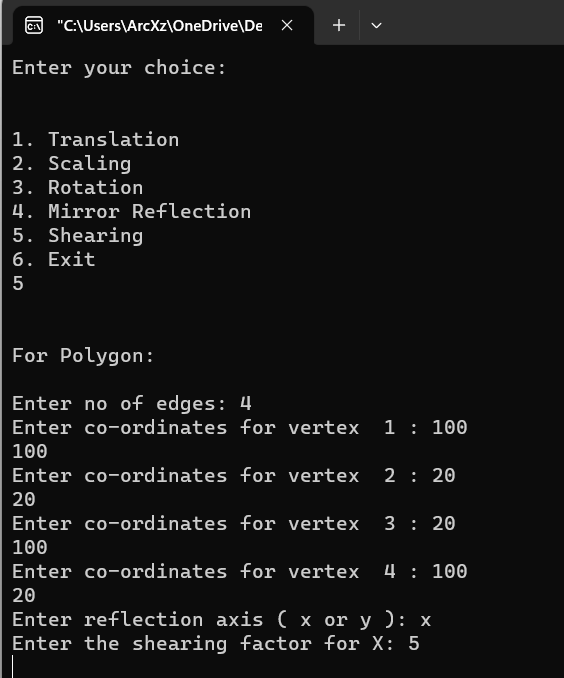
**2D REFLECTION**

****

**Chart

Description automatically generated**

**2D SHEARING**

****

**Graphical user interface, application, Teams

Description automatically generated**