2-D TRANSFORMATION

```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>
#include <graphics.h>
#define N 20
#define pi 3.142

void drawPolygon(int numOfVertices, int arr[], int color)
{
setcolor(color);
for (int i = 0; i <= numOfVertices; i += 2)
line(arr[i] + (getmaxx() / 2), (getmaxy() / 2) - arr[i + 1], arr[i + 2] +
(getmaxx() / 2), (getmaxy() / 2) - arr[i + 3]);

line(arr[i] + (getmaxx() / 2), (getmaxy() / 2) - arr[i + 1], (getmaxx() /
2) + arr[0], (getmaxy() / 2) - arr[1]);
setcolor(WHITE);
}
void drawAxes()
{
line(0, getmaxy() / 2, getmaxx(), getmaxy() / 2);
line(getmaxx() / 2, 0, getmaxx() / 2, getmaxy());
}

void Input(int array[], int numOfVertices)
{
cout << "Enter the x y value of the vertices of polygon : " << endl;
int i;
for (i = 0; i < 2 * numOfVertices; i++)
{
cin >> array[i];
}

array[i++] = array[0];
array[i] = array[1];
}
void formCoordinateMatrix(int array[], int myVerticesMatrix[][N], int
numOfVertices)
{
int k = 0;
for (int i = 0; i < numOfVertices; i++)
for (int j = 0; j < 3; j++)
{
if (j == 2)
myVerticesMatrix[i][j] = 1;
else
{
myVerticesMatrix[i][j] = array[k++];
```

```
}
}
}

void findTransformedMatrix(int newMatrix[][N], int myVerticesMatrix[][N],
float translationMatrix[][N], int numOfVertices)
{
for (int i = 0; i < numOfVertices; i++)
{
for (int j = 0; j < 3; j++)
{
newMatrix[i][j] = 0;
for (int k = 0; k < 3; k++)
newMatrix[i][j] += myVerticesMatrix[i][k] * translationMatrix[k][j];
}
}
}

void createArrayFromMatrix(int newMatrix[][N], int myArray[], int
numOfVertices)
{
int k = 0;
for (int i = 0; i < numOfVertices; i++)
{
for (int j = 0; j < 2; j++)
{
myArray[k++] = newMatrix[i][j];
}
}
myArray[k++] = myArray[0];
myArray[k] = myArray[1];
}

// TRANSLATION//
void formTranslationMatrix(float translationMatrix[][N], int tx, int ty)
{
for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++)
{
if (i == j)
translationMatrix[i][j] = 1;
else
translationMatrix[i][j] = 0;
}
translationMatrix[2][0] = tx;
translationMatrix[2][1] = ty;
}

void Translate(int myVerticesMatrix[][N], float translationMatrix[][N],
int tx, int ty, int numOfVertices)
{
```

```
int myArray[N];
int newMatrix[N][N];
formTranslationMatrix(translationMatrix, tx, ty);
findTransformedMatrix(newMatrix, myVerticesMatrix, translationMatrix,
numOfVertices);
createArrayFromMatrix(newMatrix, myArray, numOfVertices);
drawAxes();
drawPolygon(numOfVertices + 1, myArray, GREEN);
}

// END OF TRANSLATION //
//-------------------------------------------------------------------
--------
//ROTATION//

void formRotationalMatrix(float RotationMatrix[][N], float theta)
{
RotationMatrix[0][0] = (float)cos(theta);
RotationMatrix[0][1] = (float)sin(theta);
RotationMatrix[0][2] = 0;
RotationMatrix[1][0] = (float)(-sin(theta));
RotationMatrix[1][1] = (float)cos(theta);
RotationMatrix[1][2] = 0;
RotationMatrix[2][0] = 0;
RotationMatrix[2][1] = 0;
RotationMatrix[2][2] = 1;
getch();
}

void Rotate(int myVerticesMatrix[][N], float RotationMatrix[][N], float
theta, int numOfVertices)
{
int myArray[N];
int newMatrix[N][N];
formRotationalMatrix(RotationMatrix, theta);
findTransformedMatrix(newMatrix, myVerticesMatrix, RotationMatrix,
numOfVertices);
createArrayFromMatrix(newMatrix, myArray, numOfVertices);
drawAxes();
drawPolygon(numOfVertices + 1, myArray, GREEN);
}

//END OF ROTATION//
//-------------------------------------------------------------------
--------
// Scaling //
void formScalingMatrix(float ScalingMatrix[][N], int Sx, int Sy)
{
for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++)
ScalingMatrix[i][j] = 0;
```

```
ScalingMatrix[0][0] = Sx;
ScalingMatrix[1][1] = Sy;
ScalingMatrix[2][2] = 1;
}

void SCALE(int myVerticesMatrix[][N], float ScalingMatrix[][N], int Sx,
int Sy, int numOfVertices)
{
int myArray[N];
int newMatrix[N][N];
formScalingMatrix(ScalingMatrix, Sx, Sy);
findTransformedMatrix(newMatrix, myVerticesMatrix, ScalingMatrix,
numOfVertices);
createArrayFromMatrix(newMatrix, myArray, numOfVertices);
drawAxes();
drawPolygon(numOfVertices + 1, myArray, GREEN);
}

// End of Scaling //

//Reflection//

void formReflectionMatrix(float ReflectionMatrix[][N], char choice)
{
for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++)
ReflectionMatrix[i][j] = 0;

if (choice == 'X')
{
ReflectionMatrix[0][0] = 1;
ReflectionMatrix[1][1] = -1;
ReflectionMatrix[2][2] = 1;
}
else if (choice == 'Y')
{
ReflectionMatrix[0][0] = -1;
ReflectionMatrix[1][1] = 1;
ReflectionMatrix[2][2] = 1;
}
else
{
ReflectionMatrix[0][0] = -1;
ReflectionMatrix[1][1] = -1;
ReflectionMatrix[2][2] = 1;
}
}

void Reflection(int myVerticesMatrix[][N], float transformationMatrix[]
[N], char choice, int numOfVertices)
```

```
{
int myArray[N];
int newMatrix[N][N];
formReflectionMatrix(transformationMatrix, choice);
findTransformedMatrix(newMatrix, myVerticesMatrix, transformationMatrix,
numOfVertices);
createArrayFromMatrix(newMatrix, myArray, numOfVertices);
drawAxes();
drawPolygon(numOfVertices + 1, myArray, GREEN);
}

//End of Reflection//

//-------------------------------------------------------------------------
--------

//Shearing//

void formShearingMatrix(float ShearingMatrix[][N], char choice, int Shx,
int Shy)
{
for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++)
{
if (i == j)
ShearingMatrix[i][j] = 1;
else
ShearingMatrix[i][j] = 0;
}

if (choice == 'Y')
ShearingMatrix[0][1] = Shy;
else if (choice == 'X')
ShearingMatrix[1][0] = Shx;
else
{
ShearingMatrix[0][1] = Shx;
ShearingMatrix[1][0] = Shy;
}
}

void Shearing(int myVerticesMatrix[][N], float transformationMatrix[][N],
char choice, int Shx, int Shy, int numOfVertices)
{
int myArray[N];
int newMatrix[N][N];
formShearingMatrix(transformationMatrix, choice, Shx, Shy);
findTransformedMatrix(newMatrix, myVerticesMatrix, transformationMatrix,
numOfVertices);
createArrayFromMatrix(newMatrix, myArray, numOfVertices);
drawAxes();
```

```
drawPolygon(numOfVertices + 1, myArray, GREEN);
}

//End of Shearing//

int main()
{

int gd = DETECT, gm, k = 0, numOfVertices, array[N];
int myVerticesMatrix[N][N];
float translationMatrix[N][N];
float RotationMatrix[N][N];
float ReflectionMatrix[N][N];
float ShearingMatrix[N][N];
float ScalingMatrix[N][N];
int tx, ty;
int Sx, Sy;
char choice;
float thetaInRadian, theta;
//-----------------DECLARATIONS END------------------------------//
initgraph(&gd, &gm, "C:\\TC\\BGI");
cout << "Enter the number of vertices of polygon : ";
cin >> numOfVertices;
Input(array, numOfVertices);
cleardevice();
drawAxes();
drawPolygon(numOfVertices + 1, array, WHITE);

formCoordinateMatrix(array, myVerticesMatrix, numOfVertices);
int myChoice, exit = 0;
while (1)
{
cout << "-------Menu Driven------\n";
cout << "----------------------------\n";
cout << "1. Translation " << endl;
cout << "2. Rotation " << endl;
cout << "3. Scaling " << endl;
cout << "4. Shearing " << endl;
cout << "5. Reflection " << endl;
cout << "6. Exit " << endl;
cout << "Enter a choice : ";
cin >> myChoice;

switch (myChoice)
{
case 1:
{
cout << "Enter the value of tx and ty : ";
cin >> tx >> ty;
cout << "Translated in x => " << tx << "\nTranslated in y => " << ty <<
endl;
```

```
Translate(myVerticesMatrix, translationMatrix, tx, ty, numOfVertices);
drawPolygon(numOfVertices + 1, array, WHITE);
getch();
cleardevice();
break;
}
case 2:
{

cout << "ROTATION : " << endl;
cout << "Enter the angle of rotation : " << endl;
cin >> theta;
thetaInRadian = theta * ((float)pi / (float)180);
cleardevice();
cout << "ROTATED BY => " << theta << endl;
Rotate(myVerticesMatrix, RotationMatrix, thetaInRadian, numOfVertices);
drawPolygon(numOfVertices + 1, array, WHITE);
getch();
cleardevice();
break;
}
case 3:
{

cout << "Enter the value of Sx and Sy : ";
cin >> Sx >> Sy;
cout << "Scaled in x => " << Sx << "\nScaled in y => " << Sy << endl;
SCALE(myVerticesMatrix, ScalingMatrix, Sx, Sy, numOfVertices);
drawPolygon(numOfVertices + 1, array, WHITE);
getch();
cleardevice();
break;
}
case 4:
{

int Shx, Shy;
cout << "Shearing about X OR Y axes OR BOTH ? : ";
cin >> choice;
if (choice == 'X')
{
cout << "Enter Shearing Factor Along X-axis : ";
cin >> Shx;
cout << "Shearing about X axis => " << endl;
}
else if (choice == 'Y')
{
cout << "Enter Shearing Factor Along Y-axis : ";
cin >> Shy;
cout << "Shearing about Y axis => " << endl;
}
```

```
else
{
cout << "Enter Shearing Factor Along X and Y axes : ";
cin >> Shx >> Shy;
cout << "Shearing about X and Y axes => " << endl;
}
Shearing(myVerticesMatrix, ShearingMatrix, choice, Shx, Shy,
numOfVertices);
drawPolygon(numOfVertices + 1, array, WHITE);
getch();
cleardevice();
break;
}
case 5:
{
cout << "Reflection about X OR Y axes OR BOTH ? : ";
cin >> choice;
if (choice == 'X')
cout << "Reflection about X axis => " << endl;
else if (choice == 'Y')
cout << "Reflection about Y axis => " << endl;
else
cout << "Reflection about X and Y axes => " << endl;

Reflection(myVerticesMatrix, ReflectionMatrix, choice, numOfVertices);
drawPolygon(numOfVertices + 1, array, WHITE);
getch();

break;
}
case 6:
{
exit = 1;
break;
}
}
if (exit == 1)
break;
}

getch();
closegraph();
return 0;
}
```