### **Perspective projection**

#include<stdio.h>

#include<math.h>

#include<graphics.h>

main()

{

int x1,y1,x2,y2,gd,gm;

int ymax,a[4][8];

float par[4][4],b[4][8];

int i,j,k,m,n,p;

int xp, yp, zp, x, y, z;

a[0][0] = 100; a[1][0] = 100; a[2][0] = -100;

a[0][1] = 200; a[1][1] = 100; a[2][1] = -100;

a[0][2] = 200; a[1][2] = 200; a[2][2] = -100;

a[0][3] = 100; a[1][3] = 200; a[2][3] = -100;

a[0][4] = 100; a[1][4] = 100; a[2][4] = -200;

a[0][5] = 200; a[1][5] = 100; a[2][5] = -200;

a[0][6] = 200; a[1][6] = 200; a[2][6] = -200;

a[0][7] = 100; a[1][7] = 200; a[2][7] = -200;

detectgraph(&gd,&gm);

initgraph(&gd,&gm, "c:\\tc\\bgi");

ymax = getmaxy();

xp = 300; yp = 320; zp = 100;

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

{

x = a[0][j]; y = a[1][j]; z = a[2][j];

b[0][j] = xp - ( (float)( x - xp )/(z - zp)) \* (zp);

b[1][j] = yp - ( (float)( y - yp )/(z - zp)) \* (zp);

}

/\*- front plane display -\*/

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

{

x1=(int) b[0][j]; y1=(int) b[1][j];

x2=(int) b[0][j+1]; y2=(int) b[1][j+1];

line( x1,ymax-y1,x2,ymax-y2);

}

x1=(int) b[0][3]; y1=(int) b[1][3];

x2=(int) b[0][0]; y2=(int) b[1][0];

line( x1, ymax-y1, x2, ymax-y2);

/\*- back plane display -\*/

setcolor(11);

for(j=4;j<7;j++)

{

x1=(int) b[0][j]; y1=(int) b[1][j];

x2=(int) b[0][j+1]; y2=(int) b[1][j+1];

line( x1, ymax-y1, x2, ymax-y2);

}

x1=(int) b[0][7]; y1=(int) b[1][7];

x2=(int) b[0][4]; y2=(int) b[1][4];

line( x1, ymax-y1, x2, ymax-y2);

setcolor(7);

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

{

x1=(int) b[0][i]; y1=(int) b[1][i];

x2=(int) b[0][4+i]; y2=(int) b[1][4+i];

line( x1, ymax-y1, x2, ymax-y2);

}

getch();

}

# **PARALLEL PROJECTION**

# #include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int ch,x[10],y[10],z[10],a,b,c,j,xr[10],yr[10],zr[10];

int gd=DETECT,gm;

printf("\n\n parallel projection");

printf("\n\n to enter coordinates\n\n") ;

printf("\n else any other value would lead to default value \t:");

scanf("%d",&ch);

if(ch==1)

{

printf("\n follow these steps");

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

scanf("%d",&x[i],&y[i],&z[i]);

}

else

{

x[0]=y[0]=z[0]=0;

x[1]=80;y[1]=z[1]=0;

x[2]=y[2]=80;z[2]=0;

x[3]=0;y[3]=80;z[3]=0;

x[4]=0;y[4]=z[4]=80;

x[5]=y[5]=0;z[5]=80;

x[6]=80;y[6]=0;z[6]=80;

x[7]=y[7]=z[7]=80;

printf("\n default values have been set");

}

printf("\n\n now enter the projection vector \t");

scanf("%d%d%d",&a,&b,&c);

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

{

xr[i]=x[i]-(a\*z[i]/c);

yr[i]=y[i]-(b\*z[i]/c);

zr[i]=0;

}

initgraph(&gd,&gm,"");

setcolor(CYAN);

line(300,0,300,480);

line(0,240,600,240);

setcolor(YELLOW);

for(i=0,j=i+1;i<8;i++,j=(j+1)/8);

line(xr[i]+300,240-yr[i],xr[j]+300,240-yr[i]);

getch();

closegraph();

}

### **Oblique projection**

#include<stdio.h>

#include<math.h>

#include<graphics.h>

main()

{

int x1,y1,x2,y2,gd,gm;

int ymax,a[4][8];

float par[4][4],b[4][8];

int i,j,k,m,n,p;

double L1,phi;

a[0][0] = 100; a[1][0] = 100; a[2][0] = 100;

a[0][1] = 200; a[1][1] = 100; a[2][1] = 100;

a[0][2] = 200; a[1][2] = 200; a[2][2] = 100;

a[0][3] = 100; a[1][3] = 200; a[2][3] = 100;

a[0][4] = 100; a[1][4] = 100; a[2][4] = 200;

a[0][5] = 200; a[1][5] = 100; a[2][5] = 200;

a[0][6] = 200; a[1][6] = 200; a[2][6] = 200;

a[0][7] = 100; a[1][7] = 200; a[2][7] = 200;

phi = (double) (3.14\*45.0)/180 ;

L1 = 0.5;

par[0][0] = 1; par[0][1] = 0;

par[0][2] = L1\*cos(phi); par[0][3] = 0;

par[1][0] = 0; par[1][1] = 1;

par[1][2] = L1\*sin(phi); par[1][3] = 0;

par[2][0] = 0; par[2][1] = 0;

par[2][2] = 0; par[2][3] = 0;

par[3][0] = 0; par[3][1] = 0;

par[3][2] = 0; par[3][3] = 1;

m=4; n=4; p=8;

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

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

b[i][k] = 0;

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

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

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

b[i][k] += (float)par[i][j] \* a[j][k];

detectgraph(&gd,&gm);

initgraph(&gd,&gm, "c:\\tc\\bgi");

ymax = getmaxy();

/\*- front plane display -\*/

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

{

x1=(int) b[0][j]; y1=(int) b[1][j];

x2=(int) b[0][j+1]; y2=(int) b[1][j+1];

line( x1,ymax-y1,x2,ymax-y2);

}

x1=(int) b[0][3]; y1=(int) b[1][3];

x2=(int) b[0][0]; y2=(int) b[1][0];

line( x1,ymax-y1,x2,ymax-y2);

/\*- back plane display -\*/

setcolor(11);

for(j=4;j<7;j++)

{

x1=(int) b[0][j]; y1=(int) b[1][j];

x2=(int) b[0][j+1]; y2=(int) b[1][j+1];

line( x1,ymax-y1,x2,ymax-y2);

}

x1=(int) b[0][7]; y1=(int) b[1][7];

x2=(int) b[0][4]; y2=(int) b[1][4];

line( x1,ymax-y1,x2,ymax-y2);

setcolor(13);

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

{

x1=(int) b[0][i]; y1=(int) b[1][i];

x2=(int) b[0][4+i]; y2=(int) b[1][4+i];

line( x1,ymax-y1,x2,ymax-y2);

}

getch();

}

#include <cmath>

using namespace std;

typedef struct {

    float x;

    float y;

    float z;

}Point;

Point points;

float temp = 0;

void showPoint(){

    cout<<"("<<points.x<<","<<points.y<<","<<points.z<<")"<<endl;

}

void translate(float tx, float ty, float tz){

    points.x += tx;

    points.y += ty;

    points.z += tz;

    cout<<"After Translation, new point is :";

    showPoint();

}

void rotatex(float angle){

    angle = angle \* M\_PI / 180.0;

    temp = points.y;

    points.y = points.y \* cos(angle) - points.z \* sin(angle);

    points.z = temp \* sin(angle) + points.z \* cos(angle);

    cout<<"After rotation about x, new point is: ";

    showPoint();

}

void rotatey(float angle){

    angle = (angle \* M\_PI) / 180.0;

    temp = points.z;

    points.z = points.z \* cos(angle) - points.x \* sin(angle);

    points.x = temp \* sin(angle) + points.x \* cos(angle);

    cout<<"After rotation about y, new point is: ";

    showPoint();

}

void rotatez(float angle){

    angle = angle \* M\_PI / 180.0;

    temp = points.x;

    points.x = points.x \* cos(angle) - points.y \* sin(angle);

    points.y = temp \* sin(angle) + points.y \*cos(angle);

    cout<<"After rotation about z, new point is: ";

    showPoint();

}

void scale(float sf, float xf, float yf, float zf){

    points.x = points.x \* sf + (1 - sf) \* xf;

    points.y = points.y \* sf + (1 - sf) \* yf;

    points.z = points.z \* sf + (1 - sf) \* zf;

    cout<<"After scaling, new point is: ";

    showPoint();

}

int main()

{

    float tx = 0, ty = 0, tz = 0;

    float sf = 0, xf = 0, yf = 0, zf = 0;

    int choose;

    float angle;

    cout<<"Enter the initial point you want to transform:";

    cin>>points.x>>points.y>>points.z;

    cout<<"Choose the following: "<<endl;

    cout<<"1. Translate"<<endl;

    cout<<"2. Rotate about X axis"<<endl;

    cout<<"3. Rotate about Y axis"<<endl;

    cout<<"4. Rotate about Z axis"<<endl;

    cout<<"5. Scale"<<endl;

    cin>>choose;

    switch(choose){

        case 1:

            cout<<"Enter the value of tx, ty and tz: ";

            cin>>tx>>ty>>tz;

            translate(tx, ty, tz);

            break;

        case 2:

            cout<<"Enter the angle: ";

            cin>>angle;

            rotatex(angle);

            break;

        case 3:

            cout<<"Enter the angle: ";

            cin>>angle;

            rotatey(angle);

            break;

        case 4:

            cout<<"Enter the angle: ";

            cin>>angle;

            rotatez(angle);

            break;

        case 5:

            cout<<"Enter the value of sf, xf, yf and zf: ";

            cin>>sf>>xf>>yf>>zf;

            scale(sf, xf, yf, zf);

            break;

        default:

            break;

    }    return 0;}

#include<stdio.h>

#include<graphics.h>

// Matrix Multiplication to find new Coordinates.

// s[][] is scaling matrix. p[][] is to store

// points that needs to be scaled.

// p[0][0] is x coordinate of point.

// p[1][0] is y coordinate of given point.

void findNewCoordinate(int s[][2], int p[][1])

{

    int temp[2][1] = { 0 };

    for (int i = 0; i < 2; i++)

        for (int j = 0; j < 1; j++)

            for (int k = 0; k < 2; k++)

                temp[i][j] += (s[i][k] \* p[k][j]);

    p[0][0] = temp[0][0];

    p[1][0] = temp[1][0];

}

// Scaling the Polygon

void scale(int x[], int y[], int sx, int sy)

{

    // Triangle before Scaling

    line(x[0], y[0], x[1], y[1]);

    line(x[1], y[1], x[2], y[2]);

    line(x[2], y[2], x[0], y[0]);

    // Initializing the Scaling Matrix.

    int s[2][2] = { sx, 0, 0, sy };

    int p[2][1];

    // Scaling the triangle

    for (int i = 0; i < 3; i++)

    {

        p[0][0] = x[i];

        p[1][0] = y[i];

        findNewCoordinate(s, p);

        x[i] = p[0][0];

        y[i] = p[1][0];

    }

    // Triangle after Scaling

    line(x[0], y[0], x[1], y[1]);

    line(x[1], y[1], x[2], y[2]);

    line(x[2], y[2], x[0], y[0]);

}

// Driven Program

int main()

{

    int x[] = { 100, 200, 300 };

    int y[] = { 200, 100, 200 };

    int sx = 2, sy = 2;

    int gd, gm;

    detectgraph(&gd, &gm);

    initgraph(&gd, &gm," ");

    scale(x, y, sx,sy);

    getch();

    return 0;

}