**Bresenham’s line.**

#include<iostream.h>

#include<graphics.h>

#include<dos.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");

cout<<"Enter co-ordinates of first point: ";

cin>>x0>>y0;

cout<<"Enter co-ordinates of second point: ";

cin>>x1>>y1;

drawline(x0, y0, x1, y1);

delay(500);

return 0;}

**mid-point circle**

#include<iostream.h>

#include<graphics.h>

#include<dos.h>

void drawcircle(int x0, int y0, int radius){

int x=radius;

int y=0;

int err=0;

while(x>=y){

putpixel(x0 + x, y0 + y, 7);

putpixel(x0 + y, y0 + x, 7);

putpixel(x0 - y, y0 + x, 7);

putpixel(x0 - x, y0 + y, 7);

putpixel(x0 - x, y0 - y, 7);

putpixel(x0 - y, y0 - x, 7);

putpixel(x0 + y, y0 - x, 7);

putpixel(x0 + x, y0 - y, 7);

if(err<=0){

y+=1;

err+=2\*y+1;}

if(err>0){

x-=1;

err-=2\*x+1;

}}}

int main(){

int gdriver=DETECT, gmode, error, x, y, r;

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

cout<<"Enter radius of circle: ";

cin>>r;

cout<<"Enter co-ordinates of center(x and y): ";

cin>>x>>y;

drawcircle(x, y, r);

delay(500);

return 0;}

**Clip a line**

#include<iostream.h>

#include<stdlib.h>

#include<math.h>

#include<graphics.h>

#include<dos.h>

typedef struct coordinate{

int x,y;

char code[4];}PT;

void drawwindow();

void drawline(PT p1,PT p2);

PT setcode(PT p);

int visibility(PT p1,PT p2);

PT resetendpt(PT p1,PT p2);

void main()

{ int gd=DETECT,v,gm;

PT p1,p2,p3,p4,ptemp;

cout<<"\nEnter x1 and y1\n";

cin>>p1.x>>p1.y;

cout<<"\nEnter x2 and y2\n";

cin>>p2.x>>p2.y;

initgraph(&gd,&gm,"c:\\turboc3\\bgi");

drawwindow();

delay(500);

drawline(p1,p2);

delay(500);

cleardevice();

delay(500);

p1=setcode(p1);

p2=setcode(p2);

v=visibility(p1,p2);

delay(500);

switch(v){

case 0: drawwindow();

delay(500);

drawline(p1,p2);

break;

case 1: drawwindow();

delay(500);

break;

case 2: p3=resetendpt(p1,p2);

p4=resetendpt(p2,p1);

drawwindow();

delay(500);

drawline(p3,p4);

break;}

delay(5000);

closegraph();}

void drawwindow(){

line(150,100,450,100);

line(450,100,450,350);

line(450,350,150,350);

line(150,350,150,100);}

void drawline(PT p1,PT p2){

line(p1.x,p1.y,p2.x,p2.y);}

PT setcode(PT p) {

PT ptemp;

if(p.y<100)

ptemp.code[0]='1';

else

ptemp.code[0]='0';

if(p.y>350)

ptemp.code[1]='1';

else

ptemp.code[1]='0';

if(p.x>450)

ptemp.code[2]='1';

else

ptemp.code[2]='0';

if(p.x<150)

ptemp.code[3]='1';

else

ptemp.code[3]='0';

ptemp.x=p.x;

ptemp.y=p.y;

return(ptemp);}

int visibility(PT p1,PT p2){

int i,flag=0;

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

if((p1.code[i]!='0') || (p2.code[i]!='0'))

flag=1;}

if(flag==0)

return(0);

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

if((p1.code[i]==p2.code[i]) && (p1.code[i]=='1'))

flag='0';}

if(flag==0)

return(1);

return(2);}

PT resetendpt(PT p1,PT p2){

PT temp;

int x,y,i;

float m,k;

if(p1.code[3]=='1')

x=150;

if(p1.code[2]=='1')

x=450;

if((p1.code[3]=='1') || (p1.code[2]=='1')){

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

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

temp.y=k;

temp.x=x;

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

temp.code[i]=p1.code[i];

if(temp.y<=350 && temp.y>=100)

return (temp);}

if(p1.code[0]=='1')

y=100;

if(p1.code[1]=='1')

y=350;

if((p1.code[0]=='1') || (p1.code[1]=='1')){

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

k=(float)p1.x+(float)(y-p1.y)/m;

temp.x=k;

temp.y=y;

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

temp.code[i]=p1.code[i];

return(temp);}

else

return(p1);}

**Clip a polygon**

#include<stdlib.h>

int main(){

int gd,gm,n,\*x,i,k=0;

//window coordinates int wx1=220,wy1=140,wx2=420,wy2=140,wx3=420,wy3=340,wx4=220,wy4=340;

int w[]={220,140,420,140,420,340,220,340,220,140};//array for drawing window

detectgraph(&gd,&gm);

initgraph(&gd,&gm,"c:\\turboc3\\bgi");

printf("Window:-");

setcolor(RED); //red colored window

drawpoly(5,w); //window drawn

printf("Enter the no. of vertices of polygon: ");

scanf("%d",&n);

x = malloc(n\*2+1);

printf("Enter the coordinates of points:\n");

k=0;

for(i=0;i<n\*2;i+=2) {

printf("(x%d,y%d): ",k,k);

scanf("%d,%d",&x[i],&x[i+1]);

k++;}

x[n\*2]=x[0];

x[n\*2+1]=x[1];

setcolor(WHITE);

drawpoly(n+1,x);

printf("\nPress a button to clip a polygon..");

getch();

setcolor(RED);

drawpoly(5,w);

setfillstyle(SOLID\_FILL,BLACK);

floodfill(2,2,RED);

gotoxy(1,1); //bringing cursor at starting position

printf("\nThis is the clipped polygon..");

getch();

cleardevice();

closegraph();

return 0;}

**2D transformations**

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

#include<stdlib.h>

void disp(int n,float c[][3]){

float maxx,maxy;

int i;

maxx=getmaxx();

maxy=getmaxy();

maxx=maxx/2;

maxy=maxy/2;

i=0;

while(i<n-1){

line(maxx+c[i][0],maxy-c[i][1],maxx+c[i+1][0],maxy-c[i+1][1]);

i++;}

i=n-1;

line(maxx+c[i][0],maxy-c[i][1],maxx+c[0][0],maxy-c[0][1]);

setcolor(15);

line(0,maxy,maxx\*2,maxy);

line(maxx,0,maxx,maxy\*2);

setcolor(WHITE);}

void mul(int n,float b[][3],float c[][3],float a[][3]){

int i,j,k;

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

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

a[i][j]=0;

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

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

for(k=0;k<3;k++) {

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

}}

void translation(int n,float c[][3],float tx,float ty){

int i;

for(i=0;i<n;i++) {

c[i][0]=c[i][0]+tx;

c[i][1]=c[i][1]+ty; }}

void scaling(int n,float c[][3],float sx,float sy){

float b[10][3],a[10][3];

int i=0,j;

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

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

b[i][j]=0;

b[0][0]=sx;

b[1][1]=sy;

b[2][2]=1;

mul(n,b,c,a);

setcolor(MAGENTA);

disp(n,a);}

void rotationanticlock(int n,float c[][3],float ra){

int i=0,j;

float b[10][3],xp,yp,a[10][3];

xp=c[0][0];

yp=c[0][1];

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

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

b[i][j]=0;

b[0][0]=b[1][1]=cos(ra\*3.14/180);

b[0][1]=sin(ra\*3.14/180);

b[1][0]=-sin(ra\*3.14/180);

b[2][0]=(-xp\*cos(ra\*3.14/180))+(yp\*sin(ra\*3.14/180))+xp;

b[2][1]=(-xp\*sin(ra\*3.14/180))-(yp\*cos(ra\*3.14/180))+yp;

b[2][2]=1;

mul(n,b,c,a);

setcolor(10);

disp(n,a);}

void rotationclock(int n,float c[][3],float ra){

int i=0,j;

float b[10][3],xp,yp,a[10][3];

xp=c[0][0];

yp=c[0][1];

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

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

b[i][j]=0;

b[0][0]=b[1][1]=cos(ra\*3.14/180);

b[0][1]=-sin(ra\*3.14/180);

b[1][0]=sin(ra\*3.14/180);

b[2][0]=(-xp\*cos(ra\*3.14/180))+(yp\*sin(ra\*3.14/180))+xp;

b[2][1]=(-xp\*sin(ra\*3.14/180))-(yp\*cos(ra\*3.14/180))+yp;

b[2][2]=1;

mul(n,b,c,a);

setcolor(10);

disp(n,a);}

void refthrx(int n,float c[][3]){

int i=0,j;

float a[10][3],b[10][3];

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

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

b[i][j]=0;

b[0][0]=1;b[0][1]=0;b[0][2]=0;

b[1][0]=0;b[1][1]=-1;b[1][2]=0;

b[2][0]=0;b[2][1]=0;b[2][2]=1;

mul(n,b,c,a);

setcolor(3);

disp(n,a);}

void refthry(int n,float c[][3]){

int i=0,j;

float b[10][3],a[10][3];

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

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

b[i][j]=0;

b[0][0]=-1;b[0][1]=0;b[0][2]=0;

b[1][0]=0;b[1][1]=1;b[1][2]=0;

b[2][0]=0;b[2][1]=0;b[2][2]=1;

mul(n,b,c,a);

setcolor(3);

disp(n,a);}

void refthrxeqtoy(int n,float c[][3]){

int i=0,j;

float b[10][3],a[10][3];

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

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

b[i][j]=0;

b[0][0]=0;b[0][1]=1;b[0][2]=0;

b[1][0]=1;b[1][1]=0;b[1][2]=0;

b[2][0]=0;b[2][1]=0;b[2][2]=1;

mul(n,b,c,a);

setcolor(3);

disp(n,a); }

void refthrxnegy(int n,float c[][3]){

int i=0,j;

float b[10][3],a[10][3];

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

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

b[i][j]=0;

b[0][0]=0;b[0][1]=-1;b[0][2]=0;

b[1][0]=-1;b[1][1]=0;b[1][2]=0;

b[2][0]=0;b[2][1]=0;b[2][2]=1;

mul(n,b,c,a);

setcolor(3);

disp(n,a); }

void refaboutorigin(int n,float c[][3]){

int i=0,j;

float b[10][3],a[10][3];

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

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

b[i][j]=0;

b[0][0]=-1;

b[1][1]=-1;

b[2][2]=1;

mul(n,b,c,a);

setcolor(3);

disp(n,a); }

void xshearwithy(int n,float c[][3],float shx){

int i=0,j;

float b[10][3],a[10][3];

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

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

b[i][j]=0;

b[0][0]=b[1][1]=b[2][2]=1;

b[1][0]=shx;

mul(n,b,c,a);

setcolor(3);

disp(n,a); }

void shearing(int n,float c[][3]){

float b[10][3],sh,a[10][3];

int i=0,ch,j;

cleardevice();

cout<<"\n\t\* \* \* MENU \* \* \*\n";

cout<<"\n\t1) X Shearing";

cout<<"\n\t2) Y Shearing";

cout<<"\n\t3) EXIT\n";

cout<<"\n\tEnter your Choice : ";

cin>>ch;

if(ch==3)

return;

cout<<"\n\tEnter the value for Shearing : ";

cin>>sh;

cleardevice();

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

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

b[i][j]=0;

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

b[i][i]=1;

switch(ch) {

case 1:

disp(n,c);

b[1][0]=sh;

break;

case 2:

disp(n,c);

b[0][1]=sh;

break;

case 3:

break;

default:

cout<<"\n\tINVALID CHOICE !!!";

break;}

mul(n,b,c,a);

setcolor(RED);

disp(n,a);}

int main(){

int gd=DETECT,gm;

initgraph(&gd,&gm,"c:\\turboc3\\bgi");

int i,j,k,cho,n,ch2,shx,shy,yref,xref,ch3,clk;

float c[10][3],tx,ty,sx,sy,ra;

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

cin>>n;

for(i=0;i<n;i++) {

cout<<"\nEnter the co-ordinates of the vertex "<<i+1<<" : ";

cin>>c[i][0]>>c[i][1];

c[i][2]=1;}

do {

cleardevice();

cout<<"\n\t\t \* \* \* MENU FOR 2D TRANFORMATIONS \* \* \*\n";

cout<<"\n\t 1) TRANSLATION";

cout<<"\n\t 2) SCALING";

cout<<"\n\t 3) ROTATION";

cout<<"\n\t 4) REFLECTION";

cout<<"\n\t 5) SHEARING";

cout<<"\n\t 6) EXIT";

cout<<"\n\n Enter your Choice :";

cin>>cho;

switch(cho){

case 1 :

cout<<"\n\tENTER THE TRANSLATION FACTOR FOR X-AXIS : \t";

cin>>tx;

cout<<"\n\tENTER THE TRANSLATION FACTOR FOR Y-AXIS : \t";

cin>>ty;

cleardevice();

setcolor(15);

disp(n,c);

translation(n,c,tx,ty);

setcolor(14);

disp(n,c);

getch();

break;

case 2 :

cout<<"\n\tENTER THE SCALING FACTOR FOR X-AXIS :\t";

cin>>sx;

cout<<"\n\tENTER THE SCALING FACTOR FOR Y-AXIS :\t";

cin>>sy;

cleardevice();

setcolor(15);

disp(n,c);

scaling(n,c,sx,sy);

getch();

break;

case 3 :

cout<<"\n1. CLOCKWISE ROTATION";

cout<<"\n2. ANTI-CLOCKWISE ROTATION\n";

cout<<"\n Enter your Choice : ";

cin>>clk;

cout<<"\n\nENTER THE ANGLE OF ROTATION (IN DEGREE) : ";

cin>>ra;

cleardevice();

switch(clk){

case 1:

setcolor(15);

rotationanticlock(n,c,ra);

disp(n,c);

getch();

break;

case 2:

setcolor(15);

rotationclock(n,c,ra);

disp(n,c);

getch();

break;

default:

cout<<"\nINVALID CHOICE !!!";

break;}

case 4 :

cout<<"\n1. REFLECTION THROUGH X-AXIS ";

cout<<"\n2. REFLECTION THROUGH Y-AXIS";

cout<<"\n3. REFLECTION THROUGH Y=X AXIS";

cout<<"\n4. REFLECTION THROUGH Y=-X AXIS";

cout<<"\n5. REFLECTION ABOUT ORIGIN";

cout<<"\n6. EXIT\n";

cout<<"\n Enter your Choice: ";

cin>>ch2;

cleardevice();

switch(ch2){

case 1:

setcolor(15);

disp(n,c);

refthrx(n,c);

break;

case 2:

setcolor(15);

disp(n,c);

refthry(n,c);

break;

case 3:

setcolor(15);

disp(n,c);

refthrxeqtoy(n,c);

break;

case 4:

setcolor(15);

disp(n,c);

refthrxnegy(n,c);

break;

case 5:

setcolor(15);

disp(n,c);

refaboutorigin(n,c);

break;

}//end ref switch

break;

case 5 :

setcolor(15);

disp(n,c);

shearing(n,c);

getch();

break;

case 6 :

exit(0);

break;

default :

cout<<"\n\t Invalid Choice ! !";

break;

}}

while(cho!=4);

getch();

closegraph();

return 0;}

**3D**

#include<iostream>

#include<graphics.h>

#include<cctype>

#include<math.h>

#include<vector>

using namespace std;

int cube[8][4];

double transform[4][4];

int cube\_t[8][4];

int result[8][4];

void multi\_matrix() {

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

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

cube\_t[i][j] = 0;

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

cube\_t[i][j] += cube[i][k] \* transform[k][j];}

}}}

void display(int max\_x, int max\_y) {

int i;

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

line(max\_x + result[i][0], max\_y - result[i][1], max\_x + result[i + 1][0], max\_y - result[i + 1][1]);}

line(max\_x + result[3][0], max\_y - result[3][1], max\_x + result[0][0], max\_y - result[0][1]);

for(i = 4; i < 7; i++) {

line(max\_x + result[i][0], max\_y - result[i][1], max\_x + result[i + 1][0], max\_y - result[i + 1][1]);}

line(max\_x + result[7][0], max\_y - result[7][1], max\_x + result[4][0], max\_y - result[4][1]);

line(max\_x + result[0][0], max\_y - result[0][1], max\_x + result[4][0], max\_y - result[4][1]);

line(max\_x + result[1][0], max\_y - result[1][1], max\_x + result[5][0], max\_y - result[5][1]);

line(max\_x + result[3][0], max\_y - result[3][1], max\_x + result[7][0], max\_y - result[7][1]);

line(max\_x + result[2][0], max\_y - result[2][1], max\_x + result[6][0], max\_y - result[6][1]);}

void scale\_down(int max\_x, int max\_y) {

max\_x = getmaxx() / 2;

max\_y = getmaxy() / 2;

setcolor(WHITE);

line(0, max\_y, max\_x \* 2, max\_y);

line(max\_x, 0, max\_x, max\_y \* 2);}

void projection\_ortho() {

int px = 1, py = 1, pz = 0;

int p\_t[4][4];

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

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

if(i == j) {

if(i == 0) {

p\_t[i][j] = px;

} else if(i == 1) {

p\_t[i][j] = py;

} else if(i == 2) {

p\_t[i][j] = pz;

} else if(i == 3) {

p\_t[i][j] = 1;

}

} else {

p\_t[i][j] = 0;

}}}

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

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

result[i][j] = 0;

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

result[i][j] += cube\_t[i][k] \* p\_t[k][j];

}}}

int gd = DETECT, gm;

initgraph(&gd, &gm, (char\*)"");

int max\_x = getmaxx() / 2, max\_y = getmaxy() / 2;

scale\_down(max\_x, max\_y);

setcolor(LIGHTBLUE);

display(max\_x, max\_y);}

void scaling(float fx, float fy, float fz) {

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

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

if(i == j) {

if(i == 0) {

transform[i][j] = fx;

} else if(i == 1) {

transform[i][j] = fy;

} else if(i == 2) {

transform[i][j] = fz;

} else if(i == 3) {

transform[i][j] = 1;

}

} else {

transform[i][j] = 0;

}}}

multi\_matrix();}

void scaling\_overall(float f) {

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

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

if(i == j && i == 3) {

transform[i][j] = f;

} else if(i == j && i != 3) {

transform[i][j] = 1;

} else {

transform[i][j] = 0;

}}}

multi\_matrix();

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

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

cube\_t[i][j] /= f;

}}}

void rotationX(int theta, int flag) {

if(flag = 0) {

theta = - theta;}

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

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

if(i == j && (i == 0 || i == 3)) {

transform[i][j] = 1;

} else if((i == 1 && j == 1) || (i == 2 && j == 2)) {

transform[i][j] = cos((double)(theta \* 0.0174533));

} else if(i == 2 && j == 1) {

transform[i][j] = - sin((double)(theta \* 0.0174533));

} else if(i == 1 && j == 2) {

transform[i][j] = sin((double)(theta \* 0.0174533));

} else {

transform[i][j] == 0;

}}}

multi\_matrix();}

void rotationZ(int alpha, int flag) {

if(flag = 0) {

alpha = - alpha;}

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

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

if(i == j && (i == 2 || i == 3)) {

transform[i][j] = 1;

} else if((i == 0 && j == 0) || (i == 1 && j == 1)) {

transform[i][j] = cos((double)(alpha \* 0.0174533));

} else if(i == 0 && j == 1) {

transform[i][j] = sin((double)(alpha \* 0.0174533));

} else if(i == 1 && j == 0) {

transform[i][j] = - sin((double)(alpha \* 0.0174533));

} else {

transform[i][j] = 0;

}}}

multi\_matrix();}

void rotationY(int phi, int flag) {

if(flag = 0) {

phi = - phi;}

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

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

if(i == j && (i == 1 || i == 3)) {

transform[i][j] = 1;

} else if((i == 0 && j == 0) || (i == 2 && j == 2)) {

transform[i][j] = cos((double)(phi \* 0.0174533));

} else if(i == 2 && j == 0) {

transform[i][j] = sin((double)(phi \* 0.0174533));

} else if(i == 0 && j == 2) {

transform[i][j] = -sin((double)(phi \* 0.0174533));

} else {

transform[i][j] = 0;

}}}}

void reflectionXY() {

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

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

if(i == j) {

if(i == 2) {

transform[i][j] = -1;

} else {

transform[i][j] = 1;}

} else {

transform[i][j] = 0;

}}}

multi\_matrix();}

void reflectionYZ() {

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

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

if(i == j) {

if(i == 0) {

transform[i][j] = -1;

} else {

transform[i][j] = 1;}

} else {

transform[i][j] = 0;

}}}

multi\_matrix();}

void reflectionXZ() {

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

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

if(i == j) {

if(i == 1) {

transform[i][j] = -1;

} else {

transform[i][j] = 1;}

} else {

transform[i][j] = 0;

}}}

multi\_matrix();

}

void shearing(float fx1, float fx2, float fy1, float fy2, float fz1, float fz2) {

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

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

if(i == j) {

transform[i][j] = 1;

} else if(i == 0) {

if(j == 1) {

transform[i][j] = fx1;

} else if(j == 2) {

transform[i][j] = fx2;}

} else if(i == 1) {

if(j == 0) {

transform[i][j] = fy1;

} else if(j == 2) {

transform[i][j] = fy2;}

} else if(i == 2) {

if(j == 0) {

transform[i][j] = fz1;

} else if(j == 1) {

transform[i][j] = fz2;}

} else {

transform[i][j] = 0;

}}}

multi\_matrix();

}

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

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

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

if(i == j) {

transform[i][j] = 1;

} else if(i == 2) {

if(j == 0) {

transform[i][j] = tx;

} else if(j == 1) {

transform[i][j] = ty;

} else if(j == 2) {

transform[i][j] = tz;}

} else {

transform[i][j] = 0;

}}}

multi\_matrix();}

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

char ch;

int op, op\_pro;

cout<<"\n\n===========================================";

cout<<"\nPERFORM 3D TRANSFORMATIONS ON A CUBE";

cout<<"\n=============================================";

cout<<"\n\nEnter the co-ordinates of cube:-\n";

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

cout<<"Row "<<i+1<<" : ";

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

cin>>cube[i][j];}}

cube[0][3] = 1;

cube[1][3] = 1;

cube[2][3] = 1;

cube[3][3] = 1;

cube[4][3] = 1;

cube[5][3] = 1;

cube[6][3] = 1;

cube[7][3] = 1;

do {

cout<<"\n\n---------------------------";

cout<<"\n\tMAIN MENU";

cout<<"\n-----------------------------";

cout<<"\n1. Local Scaling";

cout<<"\n2. Overall Scaling";

cout<<"\n3. Rotation about x-axis (CLOCKWISE)";

cout<<"\n4. Rotation about y-axis (CLOCKWISE)";

cout<<"\n5. Rotation about z-axis (CLOCKWISE)";

cout<<"\n6. Rotation about x-axis (ANTI-CLOCKWISE)";

cout<<"\n7. Rotation about y-axis (ANTI-CLOCKWISE)";

cout<<"\n8. Rotation about z-axis (ANTI-CLOCKWISE)";

cout<<"\n9. Reflection relative to XY plane";

cout<<"\n10. Reflection relative to YZ plane";

cout<<"\n11. Reflection relative to XZ plane";

cout<<"\n12. Shearing";

cout<<"\n13. Translation";

cout<<"\nEnter choice: ";

cin>>op;

switch(op) {

case 1: {

float fx, fy, fz;

cout<<"\nYOU ARE PERFORMING LOCAL SCALING\n";

cout<<"Enter the scaling factors:-\n";

cout<<"fx: ";

cin>>fx;

cout<<"fy: ";

cin>>fy;

cout<<"fz: ";

cin>>fz;

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"\n\tOrthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

scaling(fx, fy, fz);

projection\_ortho();

}}

cout<<"Scaling with factors "<<fx<<", "<<fy<<", "<<fz<<" in the x, y and z component respectively has been done.";}

break;

case 2: {

float factor;

cout<<"\nYOU ARE PERFORMING OVERALL SCALING\n";

cout<<"Enter overall scaling factor: ";

cin>>factor;

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

scaling\_overall(factor);

projection\_ortho();

}}

cout<<"Overall scaling with a factor "<<factor<<" has been done.";}

break;

case 3: {

int theta;

cout<<"\nYOU ARE PERFORMING ROTATION ABOUT X-AXIS (CLOCKWISE)\n";

cout<<"Enter the angle by which you want to rotate the cube: ";

cin>>theta;

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

rotationX(theta, 0);

projection\_ortho();

}}

cout<<"Rotation about x-axis in the clockwise direction has been done.";}

break;

case 4: {

int phi;

cout<<"\nYOU ARE PERFORMING ROTATION ABOUT Y-AXIS (CLOCKWISE)\n";

cout<<"Enter the angle by which you want to rotate the cube: ";

cin>>phi;

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

rotationY(phi, 0);

projection\_ortho();

}}

cout<<"Rotation about y-axis in the clockwise direction has been done.";}

break;

case 5: {

int alpha;

cout<<"\nYOU ARE PERFORMING ROTATION ABOUT Z-AXIS (CLOCKWISE)\n";

cout<<"Enter the angle by which you want to rotate the cube: ";

cin>>alpha;

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

rotationZ(alpha, 0);

projection\_ortho();

}}

cout<<"Rotation about z-axis in the clockwise direction has been done.";}

break;

case 6: {

int theta;

cout<<"\nYOU ARE PERFORMING ROTATION ABOUT X-AXIS (ANTI-CLOCKWISE)\n";

cout<<"Enter the angle by which you want to rotate the cube: ";

cin>>theta;

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

rotationX(theta, 1);

projection\_ortho();

}}

cout<<"Rotation about x-axis in the anti-clockwise direction has been done.";}

break;

case 7: {

int phi;

cout<<"\nYOU ARE PERFORMING ROTATION ABOUT Y-AXIS (ANTI-CLOCKWISE)";

cout<<"\nEnter the angle by which you want to rotate the cube: ";

cin>>phi;

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

rotationY(phi, 1);

projection\_ortho();

}}

cout<<"Rotation about y-axis in the anti-clockwise direction has been done.";}

break;

case 8: {

int alpha;

cout<<"\nYOU ARE PERFORMING ROTATION ABOUT Z-AXIS (ANTI-CLOCKWISE)\n";

cout<<"Enter the angle by which you want to rotate the cube: ";

cin>>alpha;

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

rotationZ(alpha, 1);

projection\_ortho();

}}

cout<<"Rotation about z-axis in the anti-clockwise direction has been done.";}

break;

case 9: {

cout<<"\nYOU ARE PERFORMING REFLECTION RELATIVE TO XY PLANE\n";

cout<<"Reflection relative to the XY plane has been done.";

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

reflectionXY();

projection\_ortho();

}}}

break;

case 10: {

cout<<"\nYOU ARE PERFORMING REFLECTION RELATIVE TO YZ PLANE\n";

cout<<"Reflection relative to the YZ plane has been done.";

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

reflectionYZ();

projection\_ortho();

}}}

break;

case 11: {

cout<<"\nYOU ARE PERFORMING REFLECTION RELATIVE TO THE XZ PLANE\n";

cout<<"Reflection relative to the XZ plane has been done.";

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

reflectionXZ();

projection\_ortho();

}}]

break;

case 12: {

int fx1, fx2, fy1, fy2, fz1, fz2;

cout<<"\nYOU ARE PERFORMING SHEARING\n";

cout<<"Enter the shearing factors:-\n";

cout<<"fx1: ";

cin>>fx1;

cout<<"fx2: ";

cin>>fx2;

cout<<"fy1: ";

cin>>fy1;

cout<<"fy2: ";

cin>>fy2;

cout<<"fz1: ";

cin>>fz1;

cout<<"fz2: ";

cin>>fz2;

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

shearing(fx1, fx2, fy1, fy2, fz1, fz2);

projection\_ortho();

}}

cout<<"Shearing with factors "<<fx1<<", "<<fx2<<", "<<fy1<<", "<<fy2<<", "<<fz1<<", "<<fz2<<" has been done.";}

break;

case 13: {

float tx, ty, tz;

cout<<"\nYOU ARE PERFORMING TRANSLATION\n";

cout<<"Enter the translation factors in the x, y and z-direction respectively:-\n";

cout<<"tx: ";

cin>>tx;

cout<<"ty: ";

cin>>ty;

cout<<"tz: ";

cin>>tz;

cout<<"\nTO DISPLAY THE TRANSFORMED FIGURE\n";

cout<<"Select the type of projection you want to perform.";

cout<<"\n\t1. Orthographic Projection";

cout<<"\nEnter choice: ";

cin>>op\_pro;

switch(op\_pro) {

case 1: {

translation(tx, ty, tz);

projection\_ortho();

}}

cout<<"Translation with factors "<<tx<<", "<<ty<<", "<<tz<<" has been done in the direction of x, y and z-axis respectively.";}

break;

default: cout<<"\nPlease enter a valid option.";};

cout<<"\nWant to return back to menu? (y/Y - \"Yes\", any other key - \"No\") : ";

cin>>ch;

}while(toupper(ch) == 'Y');

closegraph();

return 0;}

**Hermite /Bezier curve**.

#include<iostream.h>

#include<math.h>

#include<graphics.h>

#include<dos.h>

#include<conio.h>

const int size = 4;

void bezier(int x[], int y[]) {

int gr = DETECT, gm;

initgraph(&gr, &gm, (char\*)"C:\\TURBOC3\\BGI");

double put\_x, put\_y;

cout<<"\nTHE REQUIRED CURVE:";

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

putpixel(x[i], y[i], 3);

delay(1);}

for(double t = 0; t <= 1; t += 0.001) {

put\_x = pow(1 - t, 3) \* x[0] + 3 \* t \* pow(1-t, 2) \* x[1] + 3 \* pow(t, 2) \* (1 - t) \* x[2] + pow(t, 3) \* x[3];

put\_y = pow(1 - t, 3) \* y[0] + 3 \* t \* pow(1 - t, 2) \* y[1] + 3 \* pow(t, 2) \* (1 - t) \* y[2] + pow(t, 3) \* y[3];

putpixel(put\_x, put\_y, WHITE);

delay(1);

}}

void hermite(int x[], int y[]) {

int gr = DETECT, gm;

initgraph(&gr, &gm, (char\*)("C:\\TURBOC3\\BGI"));

double put\_x, put\_y;

cout<<"\nTHE REQUIRED CURVE:";

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

putpixel(x[i], y[i], 3);

delay(1);}

for(double t = 0; t <= 1; t += 0.001) {

put\_x = (2 \* pow(t, 3) - 3 \* pow(t, 2) + 1) \* x[0] + (-2 \* pow(t, 3) + 3 \* pow(t, 2)) \* x[1] + (pow(t, 3) - 2 \* pow(t, 2) + t) \* x[2] + (pow(t, 3) - pow(t, 2)) \* x[3] ;

put\_y = (2 \* pow(t, 3) - 3 \* pow(t, 2) + 1) \* y[0] + (-2 \* pow(t, 3) + 3 \* pow(t, 2)) \* y[1] + (pow(t, 3) - 2 \* pow(t, 2) + t) \* y[2] + (pow(t, 3) - pow(t, 2)) \* y[3];

putpixel(put\_x, put\_y, WHITE);

delay(1);

}}

int main() {

int x[4], y[4];

int choice;

cout<<"\n====================================================================\n";

cout<<"\t\tHERMITE/BEZIER CURVE (USING 4 CONTROL POINTS)"; cout<<"\n====================================================================";

do {

cout<<"\nChoose any curve of your choice:";

cout<<"\n1. BEZIER CURVE";

cout<<"\n2. HERMITE CURVE";

cout<<"\nEnter your choice: ";

cin>>choice;

switch(choice) {

case 1 : {

cout<<"\nEnter the control points:-\n";

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

cout<<"x"<<i<<": ";

cin>>x[i];

cout<<"y"<<i<<": ";

cin>>y[i];}

bezier(x, y);}

break;

case 2 : {

cout<<"\nEnter the control points:-\n";

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

cout<<"x"<<i<<": ";

cin>>x[i];

cout<<"y"<<i<<": ";

cin>>y[i];}

hermite(x, y);}

break;

default : cout<<"Please enter a valid input!";}

cout<<"\nDo you want to continue?";

cin>>choice;

}while(choice=='y' || choice=='Y');

getch();

closegraph();

return 1;}

**DDA**

**#include<graphics.h>**

**#include<iostream.h>**

**#include<math.h>**

**#include<dos.h>**

**void main(){**

**float x,y,x1,y1,x2,y2,dx,dy,step;**

**int i,gd=DETECT,gm;**

**initgraph(&gd,&gm,"c:\\turboc3\\bgi");**

**cout<<"Enter the value of x1 and y1: ";**

**cin>>x1>>y1;**

**cout<<"Enter the value of x2 and y2: ";**

**cin>>x2>>y2;**

**dx=abs(x2-x1);**

**dy=abs(y2-y1);**

**if(dx>=dy)**

**step=dx;**

**else**

**step=dy;**

**dx=dx/step;**

**dy=dy/step;**

**x=x1;**

**y=y1;**

**i=1;**

**while(i<=step){**

**putpixel(x,y,5);**

**x=x+dx;**

**y=y+dy;**

**i=i+1;**

**delay(500);}**

**closegraph();}**