```
// CG Concave Polygon

#include <conio.h>
#include <iostream>
#include <graphics.h>
#include <stdlib.h>
using namespace std;

class point
{
    public:
    int x,y;
};

class poly
{
    private:
        point p[20];
        int inter[20],x,y;
        int v,xmin,ymin,xmax,ymax;
    public:
        int c;
        void read();
        void calcs();
        void display();
        void ints(float);
        void sort(int);
};


void poly::read()
{
    int i;
    cout<<"\n\t SCAN_FILL ALGORITHM";
    cout<<"\n Enter the no of vertices of polygon:";
    cin>>v;
    if(v>2)
    {
        for(i=0;i<v; i++)
        {
            cout<<"\nEnter the co-ordinate no.- "<<i+1<<" : ";
            cout<<"\n\tx"<<(i+1)<<"=";
            cin>>p[i].x;
            cout<<"\n\ty"<<(i+1)<<"=";
            cin>>p[i].y;
        }
        p[i].x=p[0].x;
        p[i].y=p[0].y;
        xmin=xmax=p[0].x;
        ymin=ymax=p[0].y;
    }
    else
        cout<<"\n Enter valid no. of vertices.";
}

void poly::calcs()
{ //MAX,MIN
    for(int i=0;i<v;i++)
    {
        if(xmin>p[i].x)
        xmin=p[i].x;
        if(xmax<p[i].x)
        xmax=p[i].x;
        if(ymin>p[i].y)
        ymin=p[i].y;
        if(ymax<p[i].y)
        ymax=p[i].y;
    }
}

void poly::display()
{
    int ch1;
    char ch='y';
    float s,s2;
```

```
    do
    {
        cout<<"\n\nMENU:";
        cout<<"\n\n\t1 . Scan line Fill ";
        cout<<"\n\n\t2 . Exit ";
        cout<<"\n\nEnter your choice:";
        cin>>ch1;
        switch(ch1)
        {
            case 1:
                s=ymin+0.01;
                delay(100);
                cleardevice();
                while(s<=ymax)
                {
                    ints(s);
                    sort(s);
                    s++;
                }
                break;
            case 2:
                exit(0);
        }

        cout<<"Do you want to continue?: ";
        cin>>ch;
    }while(ch=='y' || ch=='Y');
}

void poly::ints(float z)
{
    int x1,x2,y1,y2,temp;
    c=0;
    for(int i=0;i<v;i++)
    {
        x1=p[i].x;
        y1=p[i].y;
        x2=p[i+1].x;
        y2=p[i+1].y;
        if(y2<y1)
        {
            temp=x1;
            x1=x2;
            x2=temp;
            temp=y1;
            y1=y2;
            y2=temp;
        }
        if(z<=y2&&z>=y1)
        {
            if((y1-y2)==0)
            x=x1;
            else
            {
                x=((x2-x1)*(z-y1))/(y2-y1);
                x=x+x1;
            }
            if(x<=xmax && x>=xmin)
            inter[c++]=x;
        }
    }
}

void poly::sort(int z)
{
    int temp,j,i;

        for(i=0;i<v;i++)
        {
            line(p[i].x,p[i].y,p[i+1].x,p[i+1].y);
        }
        delay(100);
        for(i=0; i<c;i+=2)
        {
            delay(100);
```

```
            line(inter[i],z,inter[i+1],z);
        }
}

int main()
{
    int cl;
    initwindow(500,600);
    cleardevice();
    poly x;
    x.read();
    x.calcs();
    cleardevice();
    cout<<"\n\tEnter the colour u want:(0-15)->"; //Selecting colour
    cin>>cl;
    setcolor(cl);
    x.display();
    closegraph();
    getch();
    return 0;
}
```

```
// CG Cohen Sutherland

#include<iostream>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void Window()
{
 line (200,200,350,200);
 line(350,200,350,350);
 line(200,200,200,350);
 line(200,350,350,350);
}
void Code(char c[4],float x,float y)
{        c[0]=(x<200)?'1':'0';
  c[1]=(x>350)?'1':'0';
  c[2]=(y<200)?'1':'0';
  c[3]=(y>350)?'1':'0';
}
void Clipping  (char c[],char d[],float &x,float &y,float m)
{
 int flag=1,i=0;
 for (i=0;i<4;i++)
 {
  if(c[i]!='0' && d[i]!='0')
  {
   flag=0;
   break;
  }
  if(flag)
  {
   if(c[0]!='0')
   {
    y=m*(200-x)+y;
    x=200;
   }
   else if(c[1]!='0')
   {
    y=m*(350-x)+y;
    x=350;
   }
   else if(c[2]!='0')
   {
    x=((200-y)/m)+x;
    y=200;
   }
   else if(c[3]!='0')
   {
    x=((350-y)/m)+x;
    y=350;
   }
  }
  if (flag==0)
   cout<<"Line lying outside";
 }
}
void main()
{
int gdriver = DETECT, gmode, errorcode;
float x1,y1,x2,y2;
float m;
char c[4],d[4];
clrscr();
initgraph(&gdriver, &gmode, "//Turboc3//bgi");
cout<<"Enter coordinates";
cin>>x1>>y1>>x2>>y2;
cout<<"Before clipping";
Window();
line(x1,y1,x2,y2);
getch();
cleardevice();
m=float((y2-y1)/(x2-x1));
Code(c,x1,y1);
Code(d,x2,y2) ;
Clipping(c,d,x1,y1,m);
```

```
Clipping(d,c,x2,y2,m);
cout<<"After Clipping";
Window();
line(x1,y1,x2,y2);
getch();
closegraph();
}
```

```
//CG Circle Triangle

#include<iostream>
#include<graphics.h>
#include<stdio.h>
void ddaAlg(int x1,int y1,int x2,int y2)
{
 int dx=x2-x1;
 int dy=y2-y1;
 int steps=dx>dy?dx:dy;
 float xInc=dx/(float)steps;
 float yInc=dy/(float)steps;
 float x=x1;
 float y=y1;
 for(int i=0;i<=steps;i++)
 {
 putpixel(x,y,14);
 x+=xInc;
 y+=yInc;
 }
}
void display(int xc,int yc,int x,int y)
{
 putpixel(xc+x, yc+y, 3);
 putpixel(xc-x, yc+y, 3);
 putpixel(xc+x, yc-y, 3);
 putpixel(xc-x, yc-y, 3);
 putpixel(xc+y, yc+x, 3);
 putpixel(xc-y, yc+x, 3);
 putpixel(xc+y, yc-x, 3);
 putpixel(xc-y, yc-x, 3);
}
void CircleB(int x1,int y1,int r)
{
 int x=0,y=r;
 int d=3-2*r;
 display(x1,y1,x,y);
 while(y>=x)
 {
 x++;
 if(d>0)
 {
 y--;
 d=d+4*(x-y)+10;
 }
 else
 {
 d=d+4*x+6;
 }
 display(x1,y1,x,y);
 }
}
int main()
{
 int gd=DETECT, gm;
 initgraph(&gd,&gm,NULL);
 CircleB(150,180,57);
 CircleB(150,180,57/2);
 ddaAlg(102,150,198,150);
 ddaAlg(102,150,150,236);
 ddaAlg(150,236,198,150);
 getch();
 closegraph();
 return 0;
}
```

```
//CG Diamond Rectangle

#include<iostream>
#include<conio.h>
#include<graphics.h>
#include<math.h>
int sign(int x)
{
 if(x<0)
 return -1;
 else if(x>0)
 return 1;
 else
 return 0;
}
void bline(int x1,int y1,int x2,int y2,int col)
{
 int dx,dy,e,x,y,i=1;
 dx=x2-x1;
 dy=y2-y1;
 x=x1;
 y=y1;
 e=2*dy-dx;
 while(i<=dx)
 {
 while(e>=0)
 {
 y++;
 e=e-2*dx;
 }
 x++;
 e=e+2*dy;
 putpixel(x,y,col);
 i++;
 }
}
void ddaline(int x1,int y1,int x2,int y2,int
col)
{
 int x,y,len,i;
 float dx,dy;
 if(x1==x2 && y1==y2)
 putpixel(x1,y1,col);
 else
 {
 dx=x2-x1;
 dy=y2-y1;
 if(dx>dy)
 len=dx;
 else
 len=dy;
 dx=(x2-x1)/len;
 dy=(y2-y1)/len;
 x=x1+0.5*sign(dx);
 y=y1+0.5*sign(dy);
 i=1;
 while(i<len)
 {
 putpixel(x,y,col);
 x=x+dx;
 y=y+dy;
 i++;
 }
 }
}
int main()
{
 int ch,col,x1,x2,y1,y2;
 int gd=DETECT,gm;
 initgraph(&gd,&gm,"c:\\turboc3\\bgi");
 setbkcolor(WHITE);
 ddaline(50,50,50,200,2); //left vert
 ddaline(50,50,350,50,4); //up horizontal
 ddaline(350,50,350,200,6); //right vert
 ddaline(50,200,350,200,7); //down
```

```
horizontal
 ddaline(200,50,50,125,9); //diamond up left
 bline(50,125,200,200,12); //diamond
left,down
 ddaline(350,125,200,200,14);//diamond
down,right
 bline(200,50,350,125,3); //diamond right,up
 ddaline(275,87,275,163,4);//in right
 ddaline(125,87,275,87,5);//in up
 ddaline(125,87,125,163,6);//in left
 ddaline(125,163,275,163,2);//in down

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

```
// CG 2d Transformation

#include<iostream>
#include<stdlib.h>
#include<graphics.h>
#include<math.h>

using namespace std;

class POLYGON
{
    private:
        int p[10][10],Trans_result[10][10],Trans_matrix[10][10];
        float Rotation_result[10][10],Rotation_matrix[10][10];
        float Scaling_result[10][10],Scaling_matrix[10][10];
        float Shearing_result[10][10],Shearing_matrix[10][10];
        int Reflection_result[10][10],Reflection_matrix[10][10];

    public:
 int accept_poly(int [][10]);
 void draw_poly(int [][10],int);
 void draw_polyfloat(float [][10],int);
 void matmult(int [][10],int [][10],int,int,int,int [][10]);
 void matmultfloat(float [][10],int [][10],int,int,int,float [][10]);
 void shearing(int [][10],int);
 void scaling(int [][10],int);
 void rotation(int [][10],int);
 void translation(int [][10],int);
 void reflection(int [][10],int);
};

int POLYGON :: accept_poly(int p[][10])
{
 int i,n;
 cout<<"\n\nEnter number of vertices : ";
 cin>>n;
 for(i=0;i<n;i++)
 {
  cout<<"\n\nEnter (x,y) Co-ordinate of point P"<<i<<" : ";
  cin >> p[i][0] >> p[i][1];
  p[i][2] = 1;
 }

 for(i=0;i<n;i++)
 {
  cout<<"\n";
  for(int j=0;j<3;j++)
  {
   cout<<p[i][j]<<"\t\t";
  }
 }

 return n;
}
void POLYGON :: draw_poly(int p[][10], int n)
{
 int i,gd = DETECT,gm;
 initgraph(&gd,&gm,NULL);
 line(320,0,320,480);
 line(0,240,640,240);

 for(i=0;i<n;i++)
 {
  if(i<n-1)
  {
   line(p[i][0]+320, -p[i][1]+240, p[i+1][0]+320, -p[i+1][1]+240);

  }
  else
   line(p[i][0]+320, -p[i][1]+240, p[0][0]+320, -p[0][1]+240);
 }



}

void POLYGON :: draw_polyfloat(float p[][10], int n)
{
 int i,gd = DETECT,gm;
 initgraph(&gd,&gm,NULL);
 line(320,0,320,480);
 line(0,240,640,240);

 for(i=0;i<n;i++)
 {
  if(i<n-1)
  {
   line(p[i][0]+320, -p[i][1]+240, p[i+1][0]+320, -p[i+1][1]+240);

  }
  else
   line(p[i][0]+320, -p[i][1]+240, p[0][0]+320, -p[0][1]+240);
 }
```

```
}


void POLYGON :: translation(int p[10][10],int n)
{
 int tx,ty,i,j; int i1,j1,k1,r1,c1,c2;
        r1=n;c1=c2=3;
 cout << "\n\nEnter X-Translation tx : ";
 cin >> tx;
 cout << "\n\nEnter Y-Translation ty : ";
 cin >> ty;
 for(i=0;i<3;i++)
 for(j=0;j<3;j++)
  Trans_matrix[i][j] = 0;
 Trans_matrix[0][0] = Trans_matrix[1][1] = Trans_matrix[2][2] = 1;
 Trans_matrix[2][0] = tx;
 Trans_matrix[2][1] = ty;

 for(i1=0;i1<10;i1++)
 for(j1=0;j1<10;j1++)
  Trans_result[i1][j1] = 0;
 for(i1=0;i1<r1;i1++)
 for(j1=0;j1<c2;j1++)
 for(k1=0;k1<c1;k1++)
  Trans_result[i1][j1] = Trans_result[i1][j1]+(p[i1][k1] * Trans_matrix[k1][j1]);
 cout << "\n\nPolygon after Translation : ";
 draw_poly(Trans_result,n);
}

void POLYGON :: rotation(int p[][10],int n)
{
 float type,Ang,Sinang,Cosang;
        int i,j; int i1,j1,k1,r1,c1,c2;
        r1=n;c1=c2=3;
 cout << "\n\nEnter the angle of rotation in degrees : ";
 cin >> Ang;
 cout << "\n\n* * * * Rotation Types * * * *";
 cout << "\n\n1.Clockwise Rotation \n\n2.Anti-Clockwise Rotation ";
 cout << "\n\nEnter your choice(1-2): ";
 cin >> type;
 Ang = (Ang * 6.2832)/360;
 Sinang = sin(Ang);
 Cosang = cos(Ang);
          cout<<"Mark1";
 for(i=0;i<3;i++)
 for(j=0;j<3;j++)
  Rotation_matrix[i][j] = 0;
        cout<<"Mark2";
 Rotation_matrix[0][0] = Rotation_matrix[1][1] = Cosang;
 Rotation_matrix[0][1] = Rotation_matrix[1][0] = Sinang;
 Rotation_matrix[2][2] = 1;
 if(type == 1)
  Rotation_matrix[0][1] = -Sinang;
 else
  Rotation_matrix[1][0] = -Sinang;

        for(i1=0;i1<10;i1++)
 for(j1=0;j1<10;j1++)
  Rotation_result[i1][j1] = 0;
 for(i1=0;i1<r1;i1++)
 for(j1=0;j1<c2;j1++)
 for(k1=0;k1<c1;k1++)
  Rotation_result[i1][j1] = Rotation_result[i1][j1]+(p[i1][k1] * Rotation_matrix[k1][j1]);

 cout << "\n\nPolygon after Rotation : ";
        for(i=0;i<n;i++)
 {
  cout<<"\n";
  for(int j=0;j<3;j++)
  {
   cout<<Rotation_result[i][j]<<"\t\t";
  }
 }
 draw_polyfloat(Rotation_result,n);
}

void POLYGON :: scaling(int p[][10],int n)
{
 float Sx,Sy;
        int i,j; int i1,j1,k1,r1,c1,c2;
        r1=n;c1=c2=3;
 cout<<"\n\nEnter X-Scaling Sx : ";
 cin>>Sx;
 cout<<"\n\nEnter Y-Scaling Sy : ";
 cin>>Sy;

 for(i=0;i<3;i++)
 {
  for(j=0;j<3;j++)
  {
   Scaling_matrix[i][j] = 0;
  }
 }
```

```
 Scaling_matrix[0][0] = Sx;
 Scaling_matrix[0][1] = 0;
 Scaling_matrix[0][2] = 0;
 Scaling_matrix[1][0] = 0;
 Scaling_matrix[1][1] = Sy;
 Scaling_matrix[1][2] = 0;
 Scaling_matrix[2][0] = 0;
 Scaling_matrix[2][1] = 0;
 Scaling_matrix[2][2] = 1;

        for(i1=0;i1<10;i1++)
 for(j1=0;j1<10;j1++)
  Scaling_result[i1][j1] = 0;
 for(i1=0;i1<r1;i1++)
 for(j1=0;j1<c2;j1++)
 for(k1=0;k1<c1;k1++)
  Scaling_result[i1][j1] = Scaling_result[i1][j1]+(p[i1][k1] * Scaling_matrix[k1][j1]);

 cout<<"\n\nPolygon after Scaling : ";
 draw_polyfloat(Scaling_result,n);
}

void POLYGON :: shearing(int p[][10],int n)
{
 float Sx,Sy,type; int i,j;
        int i1,j1,k1,r1,c1,c2;
        r1=n;c1=c2=3;
 for(i=0;i<3;i++)
 for(j=0;j<3;j++)
 {
  if(i == j)
   Shearing_matrix[i][j] = 1;
  else
   Shearing_matrix[i][j] = 0;
 }
 cout << "\n\n* * * * Shearing Types * * * *";
 cout << "\n\n1.X-Direction Shear \n\n2.Y-Direction Shear ";
 cout << "\n\nEnter your choice(1-2) : ";
 cin >> type;
 if(type == 1)
 {
  cout << "\n\nEnter X-Shear Sx : ";
  cin >> Sx;
  Shearing_matrix[1][0] = Sx;
 }
 else
 {
  cout << "\n\nEnter Y-Shear Sy : ";
  cin >> Sy;
  Shearing_matrix[0][1] = Sy;
 }


        for(i1=0;i1<10;i1++)
 for(j1=0;j1<10;j1++)
 Shearing_result[i1][j1] = 0;
 for(i1=0;i1<r1;i1++)
 for(j1=0;j1<c2;j1++)
 for(k1=0;k1<c1;k1++)
  Shearing_result[i1][j1] = Shearing_result[i1][j1]+(p[i1][k1] * Shearing_matrix[k1][j1]);

 cout << "\n\nPolygon after Shearing : ";
 draw_polyfloat(Shearing_result,n);
}

void POLYGON :: reflection(int p[][10],int n)
{
 int type,i,j;

        int i1,j1,k1,r1,c1,c2;
  r1=n;c1=c2=3;
 cout << "\n\n* * * * Reflection Types * * * *";
 cout << "\n\n1.About X-Axis \n\n2.About Y-Axis \n\n3.About Origin\n\n4.About Line y = x \n\n5.About Line y = -x \n\nEnter your choice(1-5) : ";
 cin >> type;
 for(i=0;i<3;i++)
 for(j=0;j<3;j++)
 {
   Reflection_matrix[i][j] = 0;
 }
 switch(type)
 {
  case 1:
   Reflection_matrix[0][0] = 1;
                        Reflection_matrix[1][1] = -1;
                        Reflection_matrix[2][2] = 1;
   break;
  case 2:
   Reflection_matrix[0][0] = -1;
                        Reflection_matrix[1][1] = 1;
                        Reflection_matrix[2][2] = 1;
   break;
  case 3:
   Reflection_matrix[0][0] = -1;
```

```
                        Reflection_matrix[1][1] = -1;
                        Reflection_matrix[2][2] = 1;
   break;
  case 4:
   Reflection_matrix[0][1] = 1;
   Reflection_matrix[1][0] = 1;
   Reflection_matrix[2][2] = 1;
   break;
  case 5:
   Reflection_matrix[0][1] = -1;
   Reflection_matrix[1][0] = -1;
   Reflection_matrix[2][2] = 1;
   break;
 }

       for(i1=0;i1<10;i1++)
 for(j1=0;j1<10;j1++)
  Reflection_result[i1][j1] = 0;
 for(i1=0;i1<r1;i1++)
 for(j1=0;j1<c2;j1++)
 for(k1=0;k1<c1;k1++)
  Reflection_result[i1][j1] = Reflection_result[i1][j1]+(p[i1][k1] * Reflection_matrix[k1][j1]);

 cout << "\n\n\t\tPolygon after Reflection : ";
//cout << "\n\n\t\tPolygon after Rotationâ ¦";
       for(i=0;i<n;i++)
 {
  cout<<"\n";
  for(int j=0;j<3;j++)
  {
   cout<<Reflection_result[i][j]<<"\t\t";
  }
 }
 draw_poly(Reflection_result,n);
//closegraph();
}



int main()
{
 int ch,n,p[10][10];
 POLYGON p1;
 cout<<"\n\n* * * * 2-D TRANSFORMATION * * * *";
 n= p1.accept_poly(p);

 cout <<"\n\nOriginal Polygon : ";
 p1.draw_poly(p,n);
 do
 {

               int ch;
        cout<<"\n\n* * * * 2-D TRANSFORMATION * * * *";
  cout<<"\n\n1.Translation \n\n2.Scaling \n\n3.Rotation \
        \n\n4.Reflection \n\n5.Shearing \n\n6.Exit";
        cout<<"\n\nEnter your choice(1-6) : ";
        cin>>ch;

  switch(ch)
  {
   case 1:

    p1.translation(p,n);
    break;

   case 2:

    p1.scaling(p,n);
    break;

   case 3:

    p1.rotation(p,n);
    break;

   case 4:

    p1.reflection(p,n);
    break;

   case 5:

    p1.shearing(p,n);
    break;

   case 6:
    exit(0);
  }
 }while(1);
 return 0;
}
```

```
//CG Snowflake Koch curve

#include <iostream>
#include <math.h>
#include <graphics.h>
using namespace std;
class kochCurve
{
public:
void koch(int it,int x1,int y1,int x5,int y5)
{
int x2,y2,x3,y3,x4,y4;
int dx,dy;
if (it==0)
{
line(x1,y1,x5,y5);
}
else
{
delay(10);
dx=(x5-x1)/3;
dy=(y5-y1)/3;
x2=x1+dx;
y2=y1+dy;
x3=(int)(0.5*(x1+x5)+sqrt(3)*(y1-y5)/6);
y3=(int)(0.5*(y1+y5)+sqrt(3)*(x5-x1)/6);
x4=2*dx+x1;
y4=2*dy+y1;
koch(it-1,x1,y1,x2,y2);
koch(it-1,x2,y2,x3,y3);
koch(it-1,x3,y3,x4,y4);
koch(it-1,x4,y4,x5,y5);
}
}
};
int main()
{
kochCurve k;
int it;
cout<<"Enter Number Of Iterations : "<<endl;
cin>>it;
int gd=DETECT,gm;
initgraph(&gd,&gm,NULL);
k.koch(it,150,20,20,280);
k.koch(it,280,280,150,20);
k.koch(it,20,280,280,280);
getch();
closegraph();
return 0;
}
```

```
//CG Hilbert Curve

#include <iostream>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>

using namespace std;

void move(int j,int h,int &x,int &y)
{
if(j==1)
y-=h;
else if(j==2)
x+=h;
else if(j==3)
y+=h;
else if(j==4)
x-=h;
lineto(x,y);
}

void hilbert(int r,int d,int l,int u,int i,int h,int &x,int &y)
{
if(i>0)
{
i--;
hilbert(d,r,u,l,i,h,x,y);
move(r,h,x,y);
hilbert(r,d,l,u,i,h,x,y);
move(d,h,x,y);
hilbert(r,d,l,u,i,h,x,y);
move(l,h,x,y);
hilbert(u,l,d,r,i,h,x,y);
}
}

int main()
{
int n,x1,y1;
int x0=50,y0=150,x,y,h=10,r=2,d=3,l=4,u=1;

cout<<"\nGive the value of n: ";
cin>>n;
x=x0;y=y0;
int gm,gd=DETECT;
initgraph(&gd,&gm,NULL);
moveto(x,y);
hilbert(r,d,l,u,n,h,x,y);
delay(10000);

closegraph();

return 0;
}
```

```
// CG 3D Cube

#include<iostream>
#include<math.h>
#include<GL/glut.h>
using namespace std;
typedef float Matrix4 [4][4];
Matrix4 theMatrix;
static GLfloat input[8][3]=
{
 {40,40,-50},{90,40,-50},{90,90,-50},{40,90,-50},
 {30,30,0},{80,30,0},{80,80,0},{30,80,0}
};
float output[8][3];
float tx,ty,tz;
float sx,sy,sz;
float angle;
int choice,choiceRot;
void setIdentityM(Matrix4 m)
{
for(int i=0;i<4;i++)
 for(int j=0;j<4;j++)
 m[i][j]=(i==j);
}
void translate(int tx,int ty,int tz)
{
for(int i=0;i<8;i++)
{
output[i][0]=input[i][0]+tx;
output[i][1]=input[i][1]+ty;
output[i][2]=input[i][2]+tz;
}
}
void scale(int sx,int sy,int sz)
{
 theMatrix[0][0]=sx;
 theMatrix[1][1]=sy;
 theMatrix[2][2]=sz;
}
void RotateX(float angle) //Parallel to x
{
angle = angle*3.142/180;
 theMatrix[1][1] = cos(angle);
theMatrix[1][2] = -sin(angle);
theMatrix[2][1] = sin(angle);
theMatrix[2][2] = cos(angle);
}
void RotateY(float angle) //parallel to y
{
angle = angle*3.14/180;
theMatrix[0][0] = cos(angle);
theMatrix[0][2] = -sin(angle);
theMatrix[2][0] = sin(angle);
theMatrix[2][2] = cos(angle);
}
void RotateZ(float angle) //parallel to z
{
angle = angle*3.14/180;
theMatrix[0][0] = cos(angle);
theMatrix[0][1] = sin(angle);
theMatrix[1][0] = -sin(angle);
theMatrix[1][1] = cos(angle);
}
void multiplyM()
{
//We Don't require 4th row and column in scaling and rotation
//[8][3]=[8][3]*[3][3] //4th not used
for(int i=0;i<8;i++)
{
 for(int j=0;j<3;j++)
 {
 output[i][j]=0;
 for(int k=0;k<3;k++)
 {
 output[i][j]=output[i][j]+input[i][k]*theMatrix[k][j];
```

```
 }
 }
}
}
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])
{
 glBegin(GL_QUADS);
 glColor3f(0.7,0.4,0.5); //behind
 glVertex3fv(a[0]);
 glVertex3fv(a[1]);
 glVertex3fv(a[2]);
 glVertex3fv(a[3]);
 glColor3f(0.8,0.2,0.4); //bottom
 glVertex3fv(a[0]);
 glVertex3fv(a[1]);
 glVertex3fv(a[5]);
 glVertex3fv(a[4]);
 glColor3f(0.3,0.6,0.7); //left
 glVertex3fv(a[0]);
 glVertex3fv(a[4]);
 glVertex3fv(a[7]);
 glVertex3fv(a[3]);
 glColor3f(0.2,0.8,0.2); //right
glVertex3fv(a[1]);
glVertex3fv(a[2]);
glVertex3fv(a[6]);
glVertex3fv(a[5]);
glColor3f(0.7,0.7,0.2); //up
glVertex3fv(a[2]);
glVertex3fv(a[3]);
glVertex3fv(a[7]);
glVertex3fv(a[6]);
glColor3f(1.0,0.1,0.1);
glVertex3fv(a[4]);
glVertex3fv(a[5]);
glVertex3fv(a[6]);
glVertex3fv(a[7]);
glEnd();
}
void init()
{
 glClearColor(1.0,1.0,1.0,1.0); //set backgrond 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
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
Axes();
glColor3f(1.0,0.0,0.0);
draw(input);
setIdentityM(theMatrix);
switch(choice)
{
case 1:
 translate(tx,ty,tz);
 break;
case 2:
 scale(sx,sy,sz);
multiplyM();
 break;
```

```
case 3:
 switch (choiceRot) {
 case 1:
 RotateX(angle);
 break;
 case 2: RotateY(angle);
 break;
 case 3:
 RotateZ(angle);
 break;
 default:
 break;
 }
multiplyM();
 break;
}
draw(output);
glFlush();
}
int main(int argc, char** argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
 glutInitWindowSize(1362,750);
 glutInitWindowPosition(0,0);
 glutCreateWindow("3D TRANSFORMATIONS");
 init();
 cout<<"Enter your choice number:\n1.Translation\n2.Scaling\n3.Rotation\n=>";
 cin>>choice;
 switch (choice) {
 case 1:
 cout<<"\nEnter Tx,Ty &Tz: \n";
 cin>>tx>>ty>>tz;
 break;
 case 2:
 cout<<"\nEnter Sx,Sy & Sz: \n";
 cin>>sx>>sy>>sz;
 break;
 case 3:
 cout<<"Enter your choice for Rotation about axis:\n1.parallel to X-axis."
 <<"(y& z)\n2.parallel to Y-axis.(x& z)\n3.parallel to Z-axis."
 <<"(x& y)\n =>";
 cin>>choiceRot;
 switch (choiceRot) {
 case 1:
 cout<<"\nENter Rotation angle: ";
 cin>>angle;
 break;
 case 2:
 cout<<"\nENter Rotation angle: ";
 cin>>angle;
 break;
 case 3:
 cout<<"\nENter Rotation angle: ";
 cin>>angle;
 break;
 default:
 break;
 }
 break;
 default:
 break;
 }
 glutDisplayFunc(display);
 glutMainLoop();
return 0;
}
```

```
//CG Rainman

#include<stdio.h>
#include<graphics.h>
#define ScreenWidthgetmaxx()
#define ScreenHeightgetmaxy()
#define GroundYScreenHeight*0.74
//Use *1 for full screen
int ldisp=0;
void DrawManAndUmbrella(int x,int ldisp)
{
//head
circle(x,GroundY-90,10);
line(x,GroundY-80,x,GroundY-30);
//hand
line(x,GroundY-70,x+10,GroundY-60);
line(x,GroundY-65,x+10,GroundY-55);
line(x+10,GroundY-60,x+20,GroundY-70);
line(x+10,GroundY-55,x+20,GroundY-70);
//legs
line(x,GroundY-30,x+ldisp,GroundY);
line(x,GroundY-30,x-ldisp,GroundY);
//umbrella
pieslice(x+20,GroundY-120,0,180,40);
line(x+20,GroundY-120,x+20,GroundY-70);
}
void Rain(int x)
{
inti,rx,ry;
for(i=0;i<400;i++)
{
rx=rand() % ScreenWidth;
ry=rand() % ScreenHeight;
if(ry<GroundY-4)
{
if(ry<GroundY-120 || (ry>GroundY-120 && (rx<x-20 || rx>x+60)))
line(rx,ry,rx+0.5,ry+4);
}
}
}
void main()
{
intgd=DETECT,gm,x=0;
//Change BGI directory according to yours
initgraph(&gd,&gm,â   C:\\TurboC3\\BGIâ   );
//If you fill here (0) then you will show like flashlight
while(!kbhit())
{
//Draw Ground
line(1, GroundY,ScreenWidth,GroundY);
Rain(x);
// Increase value of(ldisp+4) for Fast moving leg
ldisp=(ldisp+4)%20;
DrawManAndUmbrella(x,ldisp);
delay(75);
cleardevice();
//If insted of(x+1) you use(x+5) or Decreasing the value moving Fast
x=(x+4)%ScreenWidth;
}
closegraph();
getch();
}
```

```
//CG Sunrise and Sunset

#include<iostream>
#include<graphics.h>
#include<cstdlib>
#include<dos.h>
#include<cmath>
using namespace std;

int main()
{
    initwindow(800,500);
    int x0,y0;
    int gdriver = DETECT,gmode,errorcode;
    int xmax,ymax;

    errorcode=graphresult();

    if(errorcode!=0)
    {
        cout<<"Graphics error:"<<grapherrormsg(errorcode);
        cout<<"Press any ket to halt";
        exit(1);
    }
    int i,j;
    setbkcolor(BLUE);
    setcolor(RED);
    rectangle(0,0,getmaxx(),getmaxy());

    outtextxy(250,240,"::::PRESS ANY KEY TO CONTINUE:::::");
    while(!kbhit());
    for(i=50,j=0;i<=250,j<=250;i+=5,j+=5)
    {
        delay(120);
        cleardevice();
        if(i<=150)
        {
            setcolor(YELLOW);
            setfillstyle(1,YELLOW);
            fillellipse(i,300-j,20,20);
        }
        else
        {
            setcolor(GREEN^RED);
            setfillstyle(1,GREEN^RED);
            fillellipse(i,300-j,20,20);
        }
    }
    delay(1000);
    cleardevice();
    setcolor(RED);
    setfillstyle(1,RED);
    fillellipse(300,50,20,20);
    delay(150);

    int k,l;
    for(k=305,l=55;k<=550,l<=300;k+=5,l+=5)
    {
        delay(120);
        cleardevice();
        if(k<=450)
        {

            setcolor(GREEN^RED);
            setfillstyle(1,GREEN^RED);

            fillellipse(k,l,20,20);
        }
        else
        {
            setcolor(YELLOW);
            setfillstyle(1,YELLOW);
            fillellipse(k,l,20,20);
        }
    }
```

```
    return 0;
}
```