Techno India NJR Institute of Technology

Lab Manual

Computer Graphics (5CS4- 21)

Akhilesh Deep Arya

Department of CSE

RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

III Year-V Semester: B.Tech. Computer Science and Engineering

5CS4-21: Computer Graphics & Multimedia Lab

Credit: 1 **Max. Marks:50 (IA:30, ETE:20)**

0L+0T+2P **End Term Exam: 2 Hours**

SN	List of Experiments
1	Implementation of Line, Circle and ellipse attributes
2	To plot a point (pixel) on the screen
3	To draw a straight line using DDA Algorithm
4	Implementation of mid-point circle generating Algorithm
5	Implementation of ellipse generating Algorithm
6	Two Dimensional transformations - Translation, Rotation, Scaling, Reflection, Shear
7	Composite 2D Transformations
8	Cohen Sutherland 2D line clipping and Windowing
9	Sutherland – Hodgeman Polygon clipping Algorithm
10	Three dimensional transformations - Translation, Rotation, Scaling
11	Composite 3D transformations
12	Drawing three dimensional objects and Scenes
13	Generating Fractal images

Course Outcome:

CO NO	Cognitive Level	Course Outcome
CO1	Application	Students will be able to pixel, line polygon, circle and ellipse primitives using C language, in 32/ 64 bit compiler.
CO2	Application	Student should be able to perform 2D transformations (translation, scaling and rotation) using C programming language.
CO3	Application	Student should be able to perform 3D transformation using C programming language.
CO4	Application	Student should be able to generate fractal images using C programming language.

Mapping COs, POs and PSOs:

Computer Programming Lab (1FY3-24/ 2FY3-24)															
Course Outcome	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**	**PSO3**
CO1	**2**	**1**	**2**	**-**	**2**	**-**	**-**	**1**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
CO2	**2**	**1**	**2**	**-**	**2**	**-**	**-**	**1**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
CO3	**2**	**1**	**2**	**-**	**2**	**-**	**-**	**1**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
CO4	**2**	**1**	**2**	**-**	**2**	**-**	**-**	**1**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
1: Slight (Low), 2: Moderate (Medium), 3: Substantial (high)															

Mapping Justification:

CO	PO	Justification
CO1 CO2 CO3 CO4	PO1	To write a C program student must require the knowledge of mathematics, that's why CO1 is moderately mapped with PO1.
	PO2	When a student write or design a C program it requires an analysis of problem statement but it does not requires complex analysis so CO1 is mapped with PO2 with low level.
	PO3	Students are required to design flow charts and algorithm, also to write C programs based on the algorithm that's why CO1 is moderately mapped with PO3.
	PO5	Writing C program require the use of 32/64 bit compilers such as code blocks and VS code, that's why CO1 is moderately mapped with PO1.
	PO8	Students are motivated to apply ethical practices while submitting the code for the problem statement. They are encouraged not to copy the code of others. That's why CO1 is mapped with PO8 with low level.

List of Practical's

1. Write a program to draw aline using Digital Differential Analyzer (DDA) algorithm
2. Write a program to draw aline using Bresenham's algorithm
3. Write a program to draw circle using midpoint algorithm.
4. Write a program to draw animated circles
5. Write a program to implement Flood fill
6. Write a program to implement Boundary fill
7. Write a program for 2D scaling of an object.
8. Write a program to rotate an object by a given angle about a given point
9. Write a program for translation of a single line
10. Write a C program to perform 3D transformations such as translation, rotation, scaling, reflection and shearing
11. Write a program to implementation for mandelbrot set fractals

1. WAP to draw aline using Digital Differential Analyzer (DDA) algorithm

Code:

```
# include <iostream.h>
 # include <graphics.h>
 # include    <conio.h>
 # include     <math.h>

 void dda_line(const int,const int,const int,const int);
 int main( )
{
        int driver=VGA;
        int mode=VGAHI;
        int x_1=0;
        int y_1=0;
        int x_2=0;
        int y_2=0;
        do
        {
                gotoxy(8,10);
                cout<<"Coordinates of Point-I (x1,y1) :";

                gotoxy(8,11);
                cout<<"==================================";

                gotoxy(12,13);
                cout<<"Enter the value of x1 = ";
                cin>>x_1;

                gotoxy(12,14);
                cout<<"Enter the value of y1 = ";
```

```
                cin>>y_1;

                gotoxy(8,18);
                cout<<"Coordinates of Point-II (x2,y2) :";

                gotoxy(8,19);
                cout<<"==================================";

                gotoxy(12,21);
                cout<<"Enter the value of x2 = ";
                cin>>x_2;

                gotoxy(12,22);
                cout<<"Enter the value of y2 = ";
                cin>>y_2;

                initgraph(&driver,&mode,"c:\\tc\\Bgi");

                setcolor(15);
                dda_line(x_1,y_1,x_2,y_2);

                setcolor(15);
                outtextxy(110,460,"Press <Enter> to continue or any other key to
                exit.");
                int key=int(getch( ));
                if(key!=13)
                        break;
        }while(1);

        return 0;
}
```

```
void dda_line(const int x_1,const int y_1,const int x_2,const int y_2)
{
        int color=getcolor( );

        int x1=x_1;
        int y1=y_1;

        int x2=x_2;
        int y2=y_2;

        if(x_1>x_2)
        {
                x1=x_2;
                y1=y_2;

                x2=x_1;
                y2=y_1;
        }
        float dx=(x2-x1);
        float dy=(y2-y1);

        int steps=abs(dy);

        if(abs(dx)>abs(dy))
                steps=abs(dx);

        float x_inc=(dx/(float)steps);
        float y_inc=(dy/(float)steps);

        float x=x1;
        float y=y1;
```

```
            putpixel(x,y,color);
            for(int count=1;count<=steps;count++)
            {
                  x+=x_inc;
                  y+=y_inc;
                  putpixel((int)(x+0.5),(int)(y+0.5),color);
            }
      }
```

Output:

Coordinates of Point-I (x1,y1)

==========

Enter the value of x1 =40

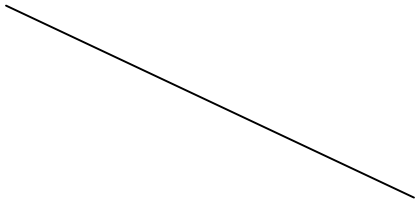
Enter the value of y1=40

Coordinates of Point-II (x2,y2)

==========

Enter the value of x2 =90

Enter the value of y2=90



2. WAP to draw aline using Bresenham's algorithm

Code:

```
# include <iostream.h>
 # include <graphics.h>
 # include    <conio.h>
 # include     <math.h>

 void bresenham_line(const int,const int,const int,const int);
 int main( )
{
        int driver=VGA;
        int mode=VGAHI;

        int x_1=0;
        int y_1=0;

        int x_2=0;
        int y_2=0;

        do
        {
                gotoxy(8,10);
                cout<<"Coordinates of Point-I (x1,y1) :";

                gotoxy(8,11);
                cout<<"================================";

                gotoxy(12,13);
                cout<<"Enter the value of x1 = ";
                cin>>x_1;
```

```
                gotoxy(12,14);
                cout<<"Enter the value of y1 = ";
                cin>>y_1;

                gotoxy(8,18);
                cout<<"Coordinates of Point-II (x2,y2) :";

                gotoxy(8,19);
                cout<<"====================================";

                gotoxy(12,21);
                cout<<"Enter the value of x2 = ";
                cin>>x_2;

                gotoxy(12,22);
                cout<<"Enter the value of y2 = ";
                cin>>y_2;
                initgraph(&driver,&mode,"c:\\tc\\Bgi");

                setcolor(15);
                bresenham_line(x_1,y_1,x_2,y_2);

                setcolor(15);
                outtextxy(110,460,"Press <Enter> to continue or any other key to
                exit.");
                int key=int(getch( ));
                if(key!=13)
                        break;
        } while(1);
        return 0;
}
```

```
void bresenham_line(const int x_1,const int y_1,const int x_2,const int y_2)
{
        int color=getcolor( );

        int x1=x_1;
        int y1=y_1;

        int x2=x_2;
        int y2=y_2;

        if(x_1>x_2)
        {
                x1=x_2;
                y1=y_2;

                x2=x_1;
                y2=y_1;
        }

        int dx=abs(x2-x1);
        int dy=abs(y2-y1);
        int inc_dec=((y2>=y1)?1:-1);

        if(dx>dy)
        {
                int two_dy=(2*dy);
                int two_dy_dx=(2*(dy-dx));
                int p=((2*dy)-dx);

                int x=x1;
                int y=y1;
```

```
                putpixel(x,y,color);
                while(x<x2)
                {
                        x++;
                        if(p<0)
                                p+=two_dy;
                        else
                        {
                                y+=inc_dec;
                                p+=two_dy_dx;
                        }
                        putpixel(x,y,color);
                }
        }
        else
        {
                int two_dx=(2*dx);
                int two_dx_dy=(2*(dx-dy));
                int p=((2*dx)-dy);

                int x=x1;
                int y=y1;

                putpixel(x,y,color);

                while(y!=y2)
                {
                        y+=inc_dec;
                        if(p<0)
                                p+=two_dx;
                        else
```

```
                    {
                        x++;
                        p+=two_dx_dy;
                    }
                    putpixel(x,y,color);
                }
            }
        }
```

Output:

Coordinates of Point-I (x1,y1)

==========

Enter the value of x1 =40

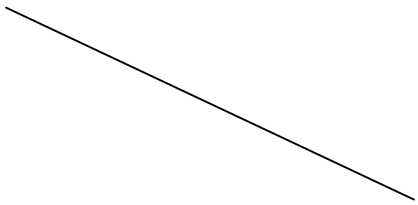
Enter the value of y1=40

Coordinates of Point-II (x2,y2)

==========

Enter the value of x2 =90

Enter the value of y2=90



3. Wap to draw circle using midpoint algorithm.

Code:

```
# include <iostream.h>
 # include <graphics.h>
 # include    <conio.h>
 # include     <math.h>

 void midpoint_circle(const int,const int,const int);

 int main( )
{
        int driver=VGA;
        int mode=VGAHI;

        int h=0;
        int k=0;
        int r=0;

        do
        {
                gotoxy(8,10);
                cout<<"Central Point of the Circle : (x,y) :";

                gotoxy(8,11);
                cout<<"=====================================";

                gotoxy(12,13);
                cout<<"Enter the value of x_center = ";
                cin>>h;
```

```
                gotoxy(12,14);
                cout<<"Enter the value of y_center = ";
                cin>>k;

                gotoxy(8,18);
                cout<<"Radius of the Circle : r :";

                gotoxy(8,19);
                cout<<"=========================================";

                gotoxy(12,21);
                cout<<"Enter the value of r = ";
                cin>>r;

                initgraph(&driver,&mode,"c:\\tc\\Bgi");

                setcolor(15);
                midpoint_circle(h,k,r);

                setcolor(15);
                outtextxy(110,460,"Press <Enter> to continue or any other key to
                exit.");

                int key=int(getch( ));

                if(key!=13)
                        break;
        }while(1);
        return 0;
}
```

```
void midpoint_circle(const int h,const int k,const int r)
{
        int color=getcolor( );
        int x=0;
        int y=r;

        int p=(1-r);

        do
        {
                putpixel((h+x),(k+y),color);
                putpixel((h+y),(k+x),color);
                putpixel((h+y),(k-x),color);
                putpixel((h+x),(k-y),color);
                putpixel((h-x),(k-y),color);
                putpixel((h-y),(k-x),color);
                putpixel((h-y),(k+x),color);
                putpixel((h-x),(k+y),color);

                x++;

                if(p<0)
                        p+=((2*x)+1);
                else
                {
                        y--;
                        p+=((2*(x-y))+1);
                }
        }while(x<=y);
}
```

Output:-

Central Point of the Circle : (x,y) :

==========

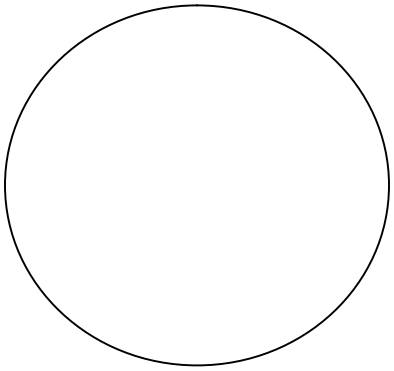
Enter the value of x_center =100

Enter the value of y_center =100

Radius of the Circle : r :

==========

Enter the value of r =50



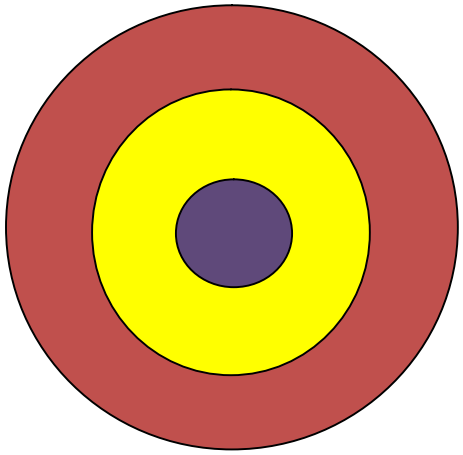
4. Write a program to draw animated circles

Code:

```
#include<stdlib.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void main()
{
        int x,y,i;
        int g=DETECT,d;
        initgraph(&g,&d,"c:\\tc\\bgi");
        cleardevice();
        x=getmaxx()/2;
        y=getmaxy()/2;
        settextstyle(TRIPLEX_FONT, HORIZ_DIR, 3);
        setbkcolor(rand());
        setcolor(4);
        outtextxy(30,100,"Press");
        outtextxy(30,130,"any");
        outtextxy(30,160,"key");
        outtextxy(30,190, "to");
        outtextxy(30,220,"Quit");
```

```
        while (!kbhit())
        {
                setcolor(rand());
                for (int i=0;i<50;i++)
                        circle(x,y,i );
                setcolor(rand());
                for (int j=70;j<120;j++)
                        circle(x,y,j);
                setcolor(rand());
                for (int k=140;k<190;k++)
                        circle(x,y,k);
                setcolor(rand());
                for (int l=210;l<230;l++)
                        circle(x,y,l);
                delay(200);
        }
        getch();
        closegraph();
}
```

Output:



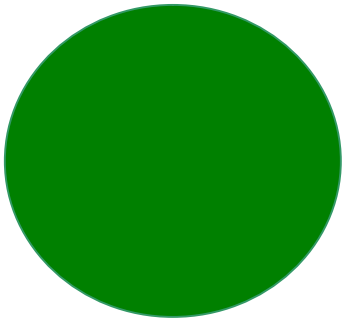
5. Write a program to implement Flood fill

Code:

```
# include <iostream.h>
# include <graphics.h>
# include    <conio.h>
# include     <math.h>

 void Flood_fill(const int,const int,const int,const int);

 int main( )
{
        int driver=VGA;
        int mode=VGAHI;

        initgraph(&driver,&mode,"c:\\tc\\Bgi");

        setcolor(15);
        circle(175,175,40);

        Flood_fill(175,175,10,0);

        getch( );
        return 0;
}

void Flood_fill(const int x,const int y, const int fill_color,const int old_color)
{
        if(getpixel(x,y)==old_color)
        {
```

```
                    putpixel(x,y,fill_color);

                    Flood_fill((x+1),y,fill_color,old_color);
                    Flood_fill((x-1),y,fill_color,old_color);
                    Flood_fill(x,(y+1),fill_color,old_color);
                    Flood_fill(x,(y-1),fill_color,old_color);
                }
        }
```

Output:



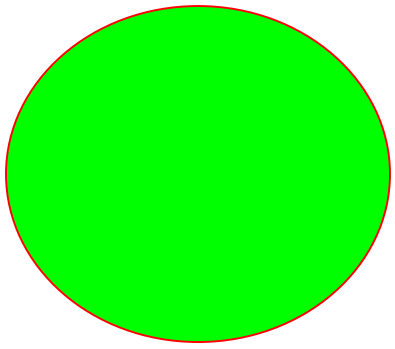
6. Write a program to implement Boundary fill

Code:

```
# include <iostream.h>
 # include <graphics.h>
 # include    <conio.h>
 # include     <math.h>

 void Boundary_fill(const int,const int,const int,const int);

 int main( )
{
        int driver=VGA;
        int mode=VGAHI;

        initgraph(&driver,&mode,"c:\\tc\\Bgi");

        setcolor(10);
        circle(175,175,40);

        Boundary_fill(175,175,5,10);

        getch( );
        return 0;
}

void Boundary_fill(const int x,const int y, const int fill_color,const int
boundary_color)
{
        if(getpixel(x,y)!=boundary_color&& getpixel(x,y)!=fill_color)
        {
```

```
                    putpixel(x,y,fill_color);

                    Boundary_fill((x+1),y,fill_color,boundary_color);
                    Boundary_fill((x-1),y,fill_color,boundary_color);
                    Boundary_fill(x,(y+1),fill_color,boundary_color);
                    Boundary_fill(x,(y-1),fill_color,boundary_color);
                }
        }
```

Output:



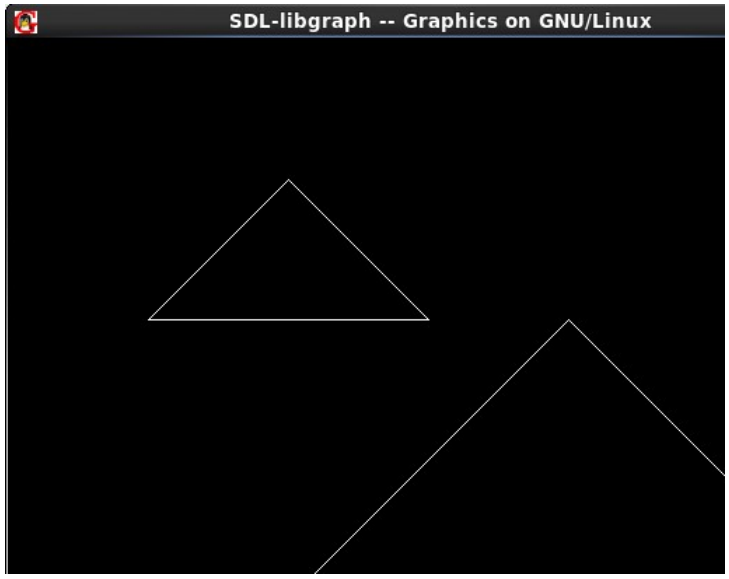
7. C program to demonstrate scaling of an object

```
#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;
}
```

Output:



8. C program to rotate an object by a given angle about a given point

```
#include <math.h>
#include <stdio.h>

// Using macros to convert degree to radian
// and call sin() and cos() as these functions
// take input in radians
#define SIN(x) sin(x * 3.141592653589 / 180)
#define COS(x) cos(x * 3.141592653589 / 180)

// To rotate an object
void rotate(float a[][2], int n, int x_pivot, int y_pivot,
                        int angle)
{
        int i = 0;
        while (i < n) {
                // Shifting the pivot point to the origin
                // and the given points accordingly
                int x_shifted = a[i][0] - x_pivot;
                int y_shifted = a[i][1] - y_pivot;

                // Calculating the rotated point co-ordinates
                // and shifting it back
                a[i][0] = x_pivot
                                + (x_shifted * COS(angle)
                                        - y_shifted * SIN(angle));
                a[i][1] = y_pivot
                                + (x_shifted * SIN(angle)
                                        + y_shifted * COS(angle));
                printf("(%f, %f) ", a[i][0], a[i][1]);
                i++;
        }
```

```
}

// Driver Code
int main()
{
        // 1st Example
        // The following figure is to be
        // rotated about (0, 0) by 90 degrees
        int size1 = 4; // No. of vertices

        // Vertex co-ordinates must be in order
        float points_list1[][2] = { { 100, 100 },
                                    { 150, 200 },
                                    { 200, 200 },
                                    { 200, 150 } };
        rotate(points_list1, size1, 0, 0, 90);

        // 2nd Example
        // The following figure is to be
        // rotated about (50, -50) by -45 degrees
        /*int size2 = 3;//No. of vertices
        float points_list2[][2] = {{100, 100}, {100, 200},
                                   {200, 200}};
        rotate(points_list2, size2, 50, -50, -45);*/
        return 0;
}
```

Output:

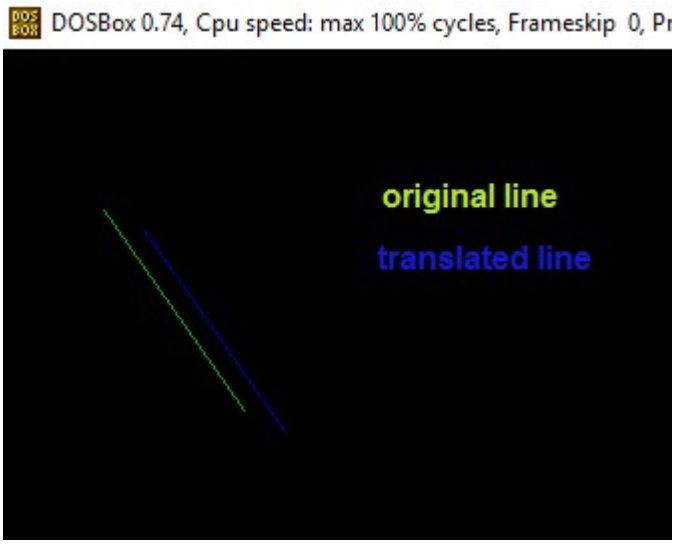
(-100, 100), (-200, 150), (-200, 200), (-150, 200)

9. cpp program for translation of a single line

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;

// function to translate line
void translateLine ( int P[][2], int T[])
{
        /* init graph and line() are used for
        representing line through graphical
        functions
        */
        int gd = DETECT, gm, errorcode;
        initgraph (&gd, &gm, "c:\\tc\\bgi");

        // drawing original line using graphics functions
        setcolor (2);
        line(P[0][0], P[0][1], P[1][0], P[1][1]);

        // calculating translated coordinates
        P[0][0] = P[0][0] + T[0];
        P[0][1] = P[0][1] + T[1];
        P[1][0] = P[1][0] + T[0];
        P[1][1] = P[1][1] + T[1];

        // drawing translated line using graphics functions
        setcolor(3);
        line(P[0][0], P[0][1], P[1][0], P[1][1]);
        closegraph();
}

// driver program
```

```
int main()
{
        int P[2][2] = {5, 8, 12, 18}; // coordinates of point
        int T[] = {2, 1}; // translation factor
        translateLine (P, T);
        return 0;
}
```

Output:



10. To write a C program to perform 3D transformations such as translation, rotation, scaling, reflection and shearing

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>

int maxx,maxy,midx,midy;

void axis()
{
        getch();
        cleardevice();
        line(midx,0,midx,maxy);
        line(0,midy,maxx,midy);
}
void main()
{
        int gd,gm,x,y,z,o,x1,x2,y1,y2;
        detectgraph(&gd,&gm);
        initgraph(&gd,&gm," ");
        setfillstyle(0,getmaxcolor());
        maxx=getmaxx();
        maxy=getmaxy();
        midx=maxx/2;
        midy=maxy/2;
        axis();
        bar3d(midx+50,midy-100,midx+60,midy-90,5,1);
        printf("Enter Translation Factor");
        scanf("%d%d%d",&x,&y,&z);
        axis();
```

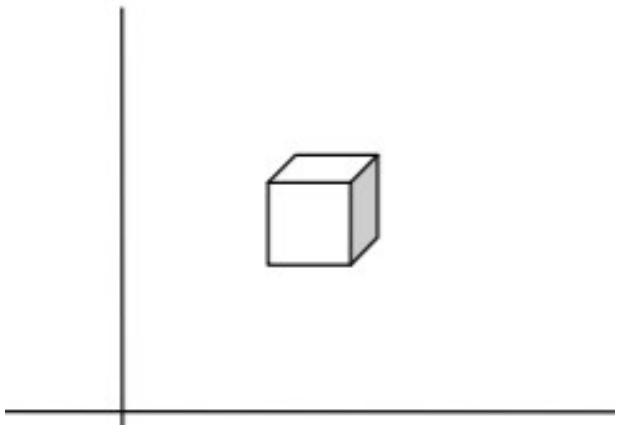
```
        printf("after translation");
        bar3d(midx+(x+50),midy-(y+100),midx+x+60,midy-(y+90),5,1);
        axis();
        bar3d(midx+50,midy+100,midx+60,midy-0,5,1);
        printf("Enter Scaling Factor");
        scanf("%d%d%d",&x,&y,&z);
        axis();
        printf("After Scaling");
        bar3d(midx+(x*50),midy-(y*100),midx+(x*60),midy-(y*90),5*z,1);
        axis();
        bar3d(midx+50,midy-100,midx+60,midy-90,5,1);
        printf("Enter Rotating Angle");
        scanf("%d",&o);
        x1=50*cos(o*3.14/180)-100*sin(o*3.14/180);
        y1=50*cos(o*3.14/180)+100*sin(o*3.14/180);
        x2=60*sin(o*3.14/180)-90*cos(o*3.14/180);
        y2=60*sin(o*3.14/180)+90*cos(o*3.14/180);
        axis();
        printf("After Rotation about Z Axis");
        bar3d(midx+x1,midy-y1,midx+x2,midy-y2,5,1);
        axis();
        printf("After Rotation about X Axis");
        bar3d(midx+50,midy-x1,midx+60,midy-x2,5,1);
        axis();
        printf("After Rotation about Y Axis");
        bar3d(midx+x1,midy-100,midx+x2,midy-90,5,1);
        getch();
        closegraph();
}
```

OUTPUT:

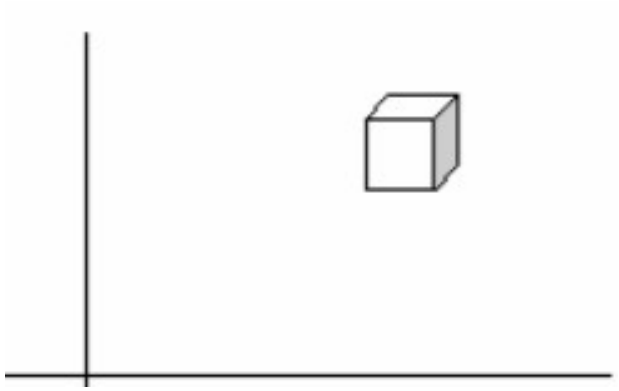
Translation

Enter Translation Factor: 50 60 70

Before Translation

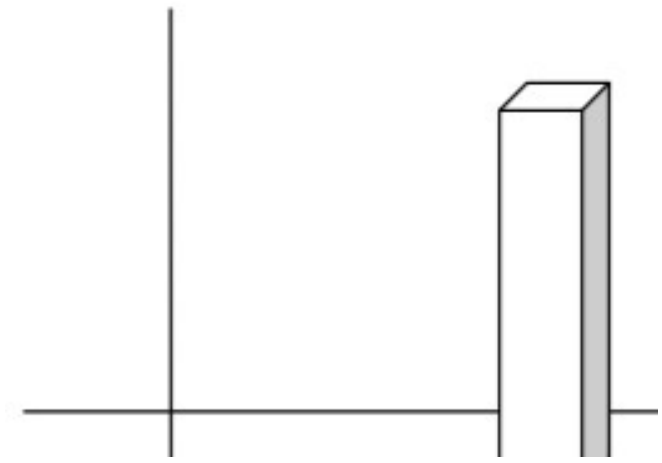


After Translation

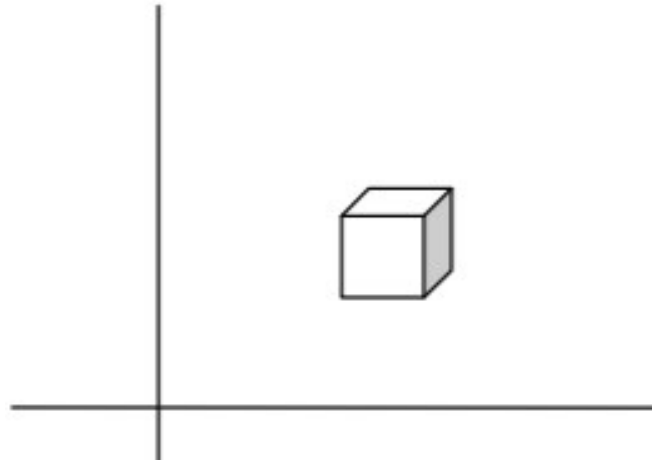


Scaling

Enter Scaling Factor: 80 90 95

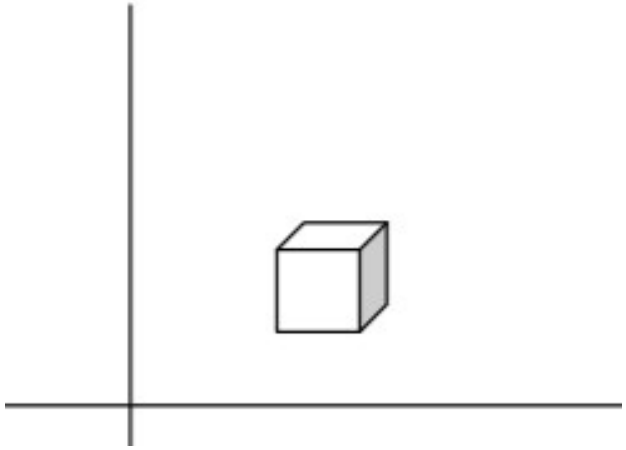


After Scaling

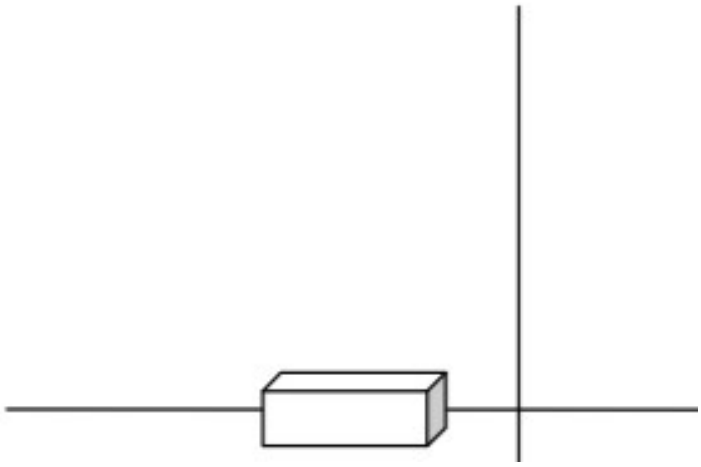


Rotation

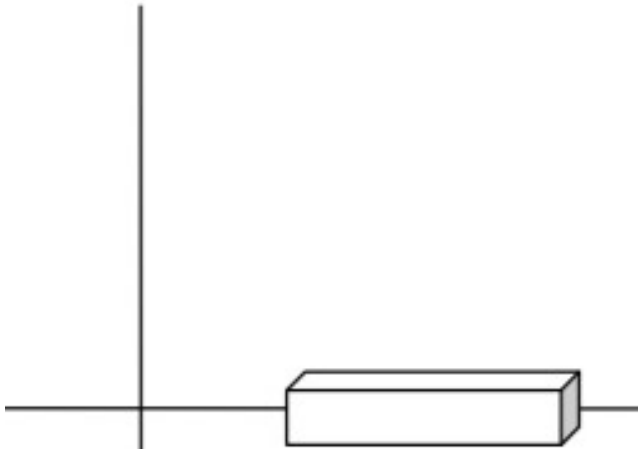
Enter Rotating Angle: 60



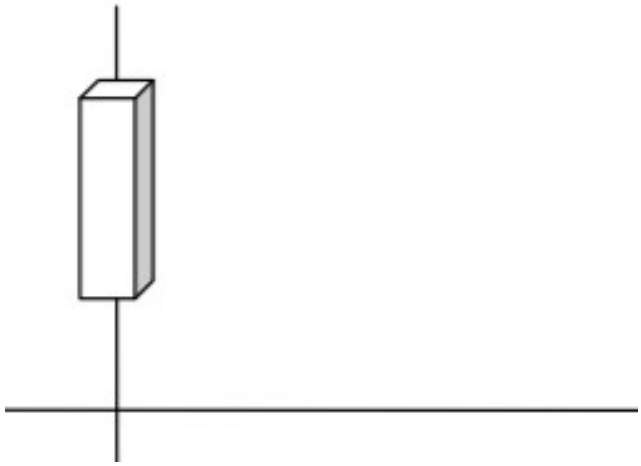
After Rotation about Z-Axis



After Rotation about X-Axis



After Rotation about Y-Axis



11. C++ implementation for mandelbrot set fractals

```
#include <graphics.h>
#include <stdio.h>
#define MAXCOUNT 30

// Function to draw mandelbrot set
void fractal(float left, float top, float xside, float yside)
{
        float xscale, yscale, zx, zy, cx, tempx, cy;
        int x, y, i, j;
        int maxx, maxy, count;

        // getting maximum value of x-axis of screen
        maxx = getmaxx();

        // getting maximum value of y-axis of screen
        maxy = getmaxy();

        // setting up the xscale and yscale
        xscale = xside / maxx;
        yscale = yside / maxy;

        // calling rectangle function
        // where required image will be seen
        rectangle(0, 0, maxx, maxy);

        // scanning every point in that rectangular area.
        // Each point represents a Complex number (x + yi).
        // Iterate that complex number
        for (y = 1; y <= maxy - 1; y++) {
                for (x = 1; x <= maxx - 1; x++)
                {
```

```
// c_real
cx = x * xscale + left;

// c_imaginary
cy = y * yscale + top;

// z_real
zx = 0;

// z_imaginary
zy = 0;
count = 0;

// Calculate whether c(c_real + c_imaginary) belongs
// to the Mandelbrot set or not and draw a pixel
// at coordinates (x, y) accordingly
// If you reach the Maximum number of iterations
// and If the distance from the origin is
// greater than 2 exit the loop
while ((zx * zx + zy * zy < 4) && (count < MAXCOUNT))
{
        // Calculate Mandelbrot function
        // z = z*z + c where z is a complex number

        // tempx = z_real*_real - z_imaginary*z_imaginary + c_real
        tempx = zx * zx - zy * zy + cx;

        // 2*z_real*z_imaginary + c_imaginary
        zy = 2 * zx * zy + cy;

        // Updating z_real = tempx
        zx = tempx;
```

```
                        // Increment count
                        count = count + 1;
                    }

                    // To display the created fractal
                    putpixel(x, y, count);
                }
            }
}

// Driver code
int main()
{
        // gm is Graphics mode which is
        // a computer display mode that
        // generates image using pixels.
        // DETECT is a macro defined in
        // "graphics.h" header file
        int gd = DETECT, gm, errorcode;

        float left, top, xside, yside;

        // setting the left, top, xside and yside
        // for the screen and image to be displayed
        left = -1.75;
        top = -0.25;
        xside = 0.25;
        yside = 0.45;
        char driver[] = "";

        // initgraph initializes the
```

```
        // graphics system by loading a
        // graphics driver from disk
        initgraph(&gd, &gm, driver);

        // Function calling
        fractal(left, top, xside, yside);

        getch();

        // closegraph function closes the
        // graphics mode and deallocates
        // all memory allocated by
        // graphics system
        closegraph();

        return 0;
}
```

Output: