MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

DEPARTMENT

OF

INFORMATION TECHNOLOGY

LAB MANUAL

16ITD12-COMPUTER GRAPHICS AND MULTIMEDIA LABORATORY

PREPARED BY

Mrs.G.Nivedhitha, M.E., Assistant Professor, Department of Information Technology

Institution Vision and Mission

Vision

To be a Centre of Excellence in Engineering, Technology and Management on par with International Standards.

Mission

- To prepare the students with high professional skills and ethical values
- To impart knowledge through best practices
- To instil a spirit of innovation through Training, Research and Development
- To undertake continuous assessment and remedial measures
- To achieve academic excellence through intellectual, emotional and social stimulation

Department Vision and Mission

Vision

To produce the Computer Science and Engineering students with the Innovative and Entrepreneur skills to face the challenges ahead

Mission

- **M1:** To impart knowledge in the state of art technologies in Computer Science and Engineering
- **M2:** To inculcate the analytical and logical skills in the field of Computer Science and Engineering
- **M3:** To produce the graduates to examine the issues and propose solutions with Ethical values

Program Educational Objectives (PEOs):

PEO1: Graduates will be able to Practice as an IT Professional in Multinational Companies

PEO2: Graduates will be able to Gain necessary skills and to pursue higher education for career growth

PEO3: Graduates will be able to Exhibit the leadership skills and ethical values in the day to day life

Program Outcomes (POs):

PO1 Engineering Knowledge: An ability to apply knowledge of basic mathematics, physical sciences and electrical engineering.

PO2 Problem Analysis: An ability to analyze a problem, interprets data, and defines electrical system requirements.

PO3 Design / Development Solutions: An ability to design, implements, and evaluate an electrical-based system, process, component to meet desired needs

PO4 Conduct Investigations of Complex Problems: An ability to design electrical and electronic systems which will provide solutions for the complex problem in the electrical domain.

PO5 Modern Tool Usage: An ability to use of modern electrical systems and electronic gadgets to provide suitable solution in the domain of electrical and electronic engineering

PO6 The Engineer and Society: An ability to give a contemporary technical and professional solutions in the practice of electrical and electronic engineering problems to meet the society needs

PO7 Environment and Sustainability: An ability to develop and use the electrical systems within realistic constraints environmental, health and safety, manufacturability, and sustainability considerations

PO8 Ethics: An Ability to understanding of professional, ethical, legal, security and social issues and responsibilities.

PO9 Individual and Team work: An ability to function effectively on teams and individually to accomplish a common goal.

PO10 Communication: An ability to communicate effectively to the higher authorities and staffs in the department.

PO11 Project Management and Finance: An ability to demonstrate leadership and managerial characteristics

PO12 Lifelong Learning: Recognize the need for and an ability to engage in life-long learning in electrical and electronics system.

Program Specific Outcomes (PSOs):

PSO1: Graduates should be able to design and analyze the algorithms to develop an Intelligent Systems

PSO2: Graduates should be able to apply the acquired skills to provide efficient solutions for real time problems

PSO3: Graduates should be able to exhibit an understanding of System Architecture, Networking and Information Security

Do’s and Dont’s in the Laboratory:

DO’s

- Know the location of the fire extinguisher and the first aid box and how to use them in case of an emergency.
- Read and understand how to carry out an activity thoroughly before coming to the laboratory.
- Use hand sanitizer and face mask before enter in the Laboratory
- Report fires or accidents to your lecturer/laboratory technician immediately.
- Report any broken plugs or exposed electrical wires to your lecturer/laboratory technician immediately.

DON’Ts

- Do not eat or drink in the laboratory.
- Avoid stepping on electrical wires or any other computer cables.
- Do not open the system unit casing or monitor casing particularly when the power is turned on. Some internal components hold electric voltages of up to 30000 volts, which can be fatal.
- Do not insert metal objects such as clips, pins and needles into the computer casings. They may cause fire.
- Do not remove anything from the computer laboratory without permission.
- Do not touch, connect or disconnect any plug or cable without your lecturer/laboratory technician’s permission.
- Do not misbehave in the computer laboratory.

LIST OF PROGRAMS

1. Implementation of DDA and Bresenhams Line Algorithms for all slopes.
2. Implementation of Midpoint Circle Algorithm.
3. 2D Geometric Transformations – Translation, Rotation, Scaling, Reflection, Shearing.
4. Cohen - Sutherland Line Clipping Algorithm. Implement the exercises from 5 to 7 using OpenGL.
5. 3D Transformations - Translation, Rotation, Scaling.
6. 3D Projections – Parallel, Perspective.
7. Creating 3D Scenes.
8. Compression Algorithms – To implement text and image compression algorithms
9. Image Editing and Manipulation - Basic operations on image using any image editing Software, creating gif animated images, Image optimization.
10. 2D Animation – To create interactive animation using any authoring tool.

1.IMPLEMENTATION OF DDA BRESENHAMS LINE ALGORITHM FOR SLOPES

AIM:

To write a C program for draw a line using DDA line Algorithm.

OBJECTIVES:

To understand DDA commands.

ALGORITHM:

1. Start the program.
2. Enter the starting and ending point of the line.
3. Call the initgraph() function.
4. Invoke the function draw, and calculate the absolute value of dx and dy and check if abs(dx)>abs(dy).
5. If true assign step size as abs(dx), or else assign as abs(dy).
6. Calculate the x in c, y in c and plot the start point use the loop, until k is less than or equal to step size.
7. Calculate the x and y for each steps and plot the corresponding pixels and let the line be displayed.
8. Stop the graphics driver.

Stop the program.

THEORY:

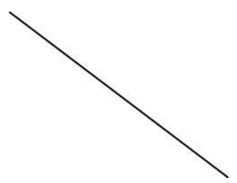
In any 2-Dimensional plane, if we connect two points (x0, y0) and (x1, y1), we get a line segment. But in the case of computer graphics, we can not directly join any two coordinate points, for that, we should calculate intermediate points' coordinates and put a pixel for each intermediate point, of the desired color with the help of functions like putpixel(x, y, K) in C, where (x,y) is our co-ordinate and K denotes some color.

PROGRAM:

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

void draw(int x1,int y1,int x2,int y2);
int main(void)
{
   int x1,y1,x2,y2;
        intgdriver=DETECT,gmode,gerror;
        printf("\n Enter the x and y value for starting point:\n");
        scanf("%d%d",&x1,&y1);
     printf("\n Enter the x and y value for ending point:\n");
     scanf("%d%d",&x2,&y2);
     clrscr();
     initgraph(&gdriver,&gmode,"E:\\TC\\BGI\\");
     draw(x1,y1,x2,y2);
     getch();
     return 0;
}

void draw(int x1,int y1,int x2,int y2)
{

     floatx,y,xinc,yinc,dx,dy;
     intk,step;
     dx=x2-x1;
     dy=y2-y1;
     if(abs(dx)>abs(dy))
          step=abs(dx);
     else
          step=abs(dy);

     xinc=dx/step;
     yinc=dy/step;
     x=x1;
     y=y1;
     putpixel(abs(x),abs(y),111);
     for(k=1;k<=step;k++)
     {
          x=x+xinc;
          y=y+yinc;
          putpixel(abs(x),abs(y),111);
     }
}
```

OUTPUT:

```
Enter the x and y value for starting point: 50 60
Enter the x and y value for ending point: 70 90
```



BRESENHAM’S ALGORITHM

AIM:

To write a program in C to draw a line using Bresenham’s algorithm.

ALGORITHM:

1. Start the program.
2. Initialize the variables.
3. Call the initgraph() function.
4. Input the two line end-points and store the left end-point in (x_0, y_0).
5. Plot the point (x_0, y_0).
6. Calculate the constantsx, y, 2Δy, and (2Δy - 2Δx) and get the first value for thedecision parameter as:

7. At each x_k along the line, starting at $k = 0$, perform the following test. If $p_k < 0$, the next point to plot is (x_k+1, y_k) and
8. Otherwise, the next point to plot is (x_k+1, y_k+1) and Repeat step 4 (Δx – 1) times.
   xyp −Δ=20*yppkk*Δ+=+21*xyppkk* −Δ+=+221
9. Stop the graphics driver.
10. Stop the program.

THEORY:

This algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly.

In this method, next pixel selected is that one who has the least distance from true line

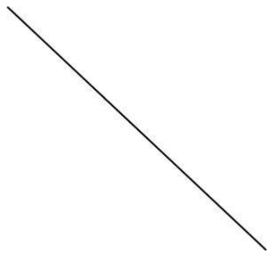
PROGRAM:

```
# include <stdio.h>
# include <conio.h>
# include <graphics.h> void main( )
{
        int x,y,x1,y1,x2,y2,p,dx,dy; intgdriver=DETECT,gmode;
        initgraph(&gdriver,&gmode,"C:\\tc\\BGI:"); printf("\nEnter the x-
        coordinate of the first point ::"); scanf("%d",&x1);
        printf("\nEnter the y-coordinate of the first point ::");
        scanf("%d",&y1);
        printf("\nEnter the x-coordinate of the second point ::");
        scanf("%d",&x2);
        printf("\nEnter the y-coordinate of the second point ::");
        scanf("%d",&y2);
        x=x1;
```

```
	y=y1; dx=x2-x1; dy=y2-y1;
	putpixel(x,y,2);
	p=(2dy-dx);
	while(x<=x2)
	{
		if(p<0)
		{
			x=x+1;
			p=2*x-dx;
		}
		else
		{
			x=x+1;
			y=y+1;
			p=p+2*dy;
		}
		putpixel(x,y,7);
	}
	getch();
	closegraph();
}
```

OUTPUT:

```
Enter the x-coordinate of the first point ::40
Enter the y-coordinate of the first point ::50
Enter the x-coordinate of the second point ::60
Enter the y-coordinate of the second point ::80
```



RESULT:

Thus the program in C to draw the line by using DDA and Bresenham's algorithm was done successfully

2. IMPLEMENTATION OF MID POINT CIRCLE ALGORITHMS

AIM:

To write a program in C to draw a circle using Bresenham’s Algorithm.

ALGORITHM:

1. Start the program.
2. Initialize the variables.
3. Call the initgraph() function.
4. Input radius r and circle centre(x_c, y_c), then set the coordinates for the first point on the circumference of a circle centred on the origin as:
5. Calculate the initial value of the decision parameter as:
6. Starting with $k = 0$ at each position x_k, perform the following test. If $p_k < 0$, the next point along the circle centred on $(0, 0)$ is (x_k+1, y_k) and rp–=450),
7. Otherwise the next point along the circle is (x_k+1, y_k-1) and
8. Determine symmetry points in the other seven octants
9. Move each calculated pixel position (x, y) onto the circular path centred at (x_c, y_c) to plot the coordinate values:
10. Repeat steps 3 to 5 until $x >= y$
11. Stop the graphics driver.
12. Stop the program.

THEORY:

The mid-point circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle.

We use the mid-point algorithm to calculate all the perimeter points of the circle in the **first octant** and then print them along with their mirror points in the other octants. This will work because a circle is symmetric about its centre.

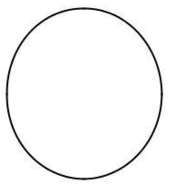
PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
voidcirclepoints(int,int);
void main( )
{
intx,y,p,r;
intgdriver=DETECT,gmode;
```

```
        initgraph(&gdriver,&gmode,"C:\\tc\\bgi:");
        clrscr();
        printf("Enter the radius");
        scanf("%d",&r);
        x=0;y=r;p=1-r;
        while(x<y)
        {
        x++;
        if(p>0)
        {
        p=p+2*(x-y)+1;
                        y--;
                }
                else
                        p=p+2*x+1;
                        circlepoints(x,y);
         }
         getch();
         closegraph();
}

voidcirclepoints(intx,int y)
{
         putpixel(x+300,y+300,8);
         putpixel(x+300,-y+300,8);
         putpixel(-x+300,y+300,8);
         putpixel(-x+300,-y+300,8);
         putpixel(y+300,x+300,8);
         putpixel(y+300,-x+300,8);
         putpixel(-y+300,x+300,8);
         putpixel(-y+300,-x+300,8);
}
```

OUTPUT:

```
Enter the radius 50
```



RESULT:

Thus the program in C to draw the circle by using Bresenham’s algorithm was done successfully.

3. 2D GEOMETRIC TRASFORMATION – TRANSLATION, ROTATION, SCALING, REFLECTION, SHEARING

AIM:

To implement the following 2D Geometric Transformations

a. Translation b. Rotation c. Scaling d. Reflection e. Shearing

ALGORITHM:

1. Get the coordinates of triangle (x1, y1, x2, y2, x3, y3).
2. Draw the original triangle.
3. Print the menu for choosing 2D Geometric Transformation.

a. If users choose translation then get the translation factors(x, y) and draw the translated triangle in the following coordinates (x1+x, y1+y, x2+x, y2+y, x3+x, y3+y).

b. (i) If user choose rotation then get the rotation angle (t) and reference point of the rotation (rx, ry).

(ii) Change the t value to t = t * (3.14 / 180) and calculate the rotated coordinates by the following formulae rx1 = rx + (x1 - rx) * cos (t) - (y1 - ry) * sin (t); ry1 = ry + (x1 - rx) * sin (t) + (y1 - ry) *cos (t);

(iii) Similarly calculate the coordinates rx2, ry2, rx3, ry3 and draw the rotated triangle in the following coordinates (rx1, ry1, rx2, ry2, rx3, ry3).

c. If user choose scaling then get the scaling factors(x, y) and draw the scaleded triangle in the following coordinates (x1*x, y1*y, x2*x, y2*y, x3*x, y3*y).

d. If user choose reflection then rotate the triangle in 1800 at (x2, y2) and draw the rotated triangle (which is reflected triangle).

e. If user choose shearing then get the shear value and draw the sheared triangle in the following coordinates (x1, y1, x2+x, y2, x3, y3).

f. (i) if user choose window-view port then draw the rectangle in window port coordinates (w1, w2, w3, w4) and draw the original triangle.

(ii) Calculate the x, y and view port coordinates by following formulae

x = (v3 - v1) / (w3 - w1);

y = (v4 - v2) / (w4 - w2);

vx1 = v1 + floor (((x1 - w1) * x) + 0.5);

vy1 = v2 + floor (((y1 - w2) * y) + 0.5);

(iii) Similarly calculate the coordinates vx2, vy2, vx3, vy3

(iv) Draw the rectangle in view port coordinates (v1, v2, v3, v4) and draw the triangle in view Port by the following coordinates (vx1, vy1, vx2, vy2, vx3, vy3)

THEORY:

Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation.

Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

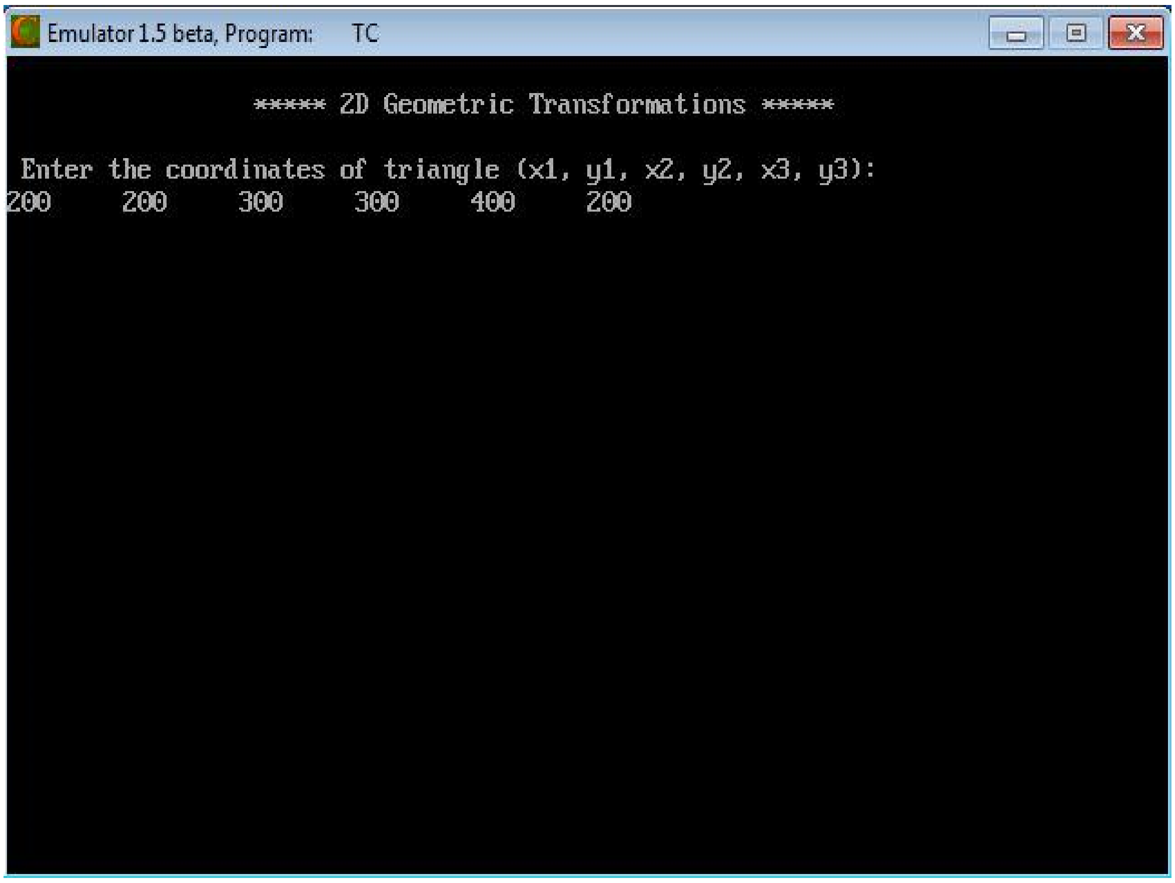
PROGRAM:

```
#include <graphics.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>
float x1, y1, x2, y2, x3, y3, x, y, rx1, ry1, rx2, ry2, rx3, ry3, t, vx1, vy1, vx2, vy2, vx3, vy3; floatw1 =
5, w2 = 5, w3 = 635, w4 = 465, v1 = 425, v2 = 75, v3 = 550, v4 = 250; intgd, gm, ch;
void original ();
void triangle (float, float, float, float, float, float); void rotate
(float, float, float); void main ()
{
        clrscr ();
        cout<< "\n\t\t ***** 2D Geometric Transformations *****";
        cout<< "\n\n Enter the coordinates of triangle (x1,y1,x2,y2,x3, y3): \n"; cin>> x1
        >> y1 >> x2 >> y2 >> x3 >> y3; original ();

        closegraph ();
        do
        {
                cout<< "\n\n Choose any one Transformation : ";
                cout<< "\n\t 1.Translation \n\t 2.Rotation \n\t 3.Scaling \n\t 4.Reflection"; cout<<
                "\n\t 5.Shearing \n\t 6.Window-Viewport"; cout<< "\n\n Enter your choice : \t";
                cin>>ch;
                switch (ch)
                {
                        case 1:
                                cout<< "\n Enter the translation factors (x,y): \t\t";
                                cin>> x >> y;
                                original ();
                                triangle (x1+x, y1+y, x2+x, y2+y, x3+x, y3+y);
                                closegraph ();
                                break;
                        case 2:
                                cout<< "\n Enter the angle of rotation : \t\t";
                                cin>> t;
                                cout<< "\n Enter the reference point of rotation (rx,ry) : \t";
                                cin>> x >> y;
                                original ();
                                rotate (t, x, y);
                                closegraph ();
                                break;
                        case 3:
                                cout<< "\n Enter the scaling factors (x,y): \t\t";
                                cin>> x >> y;
                                original ();
```

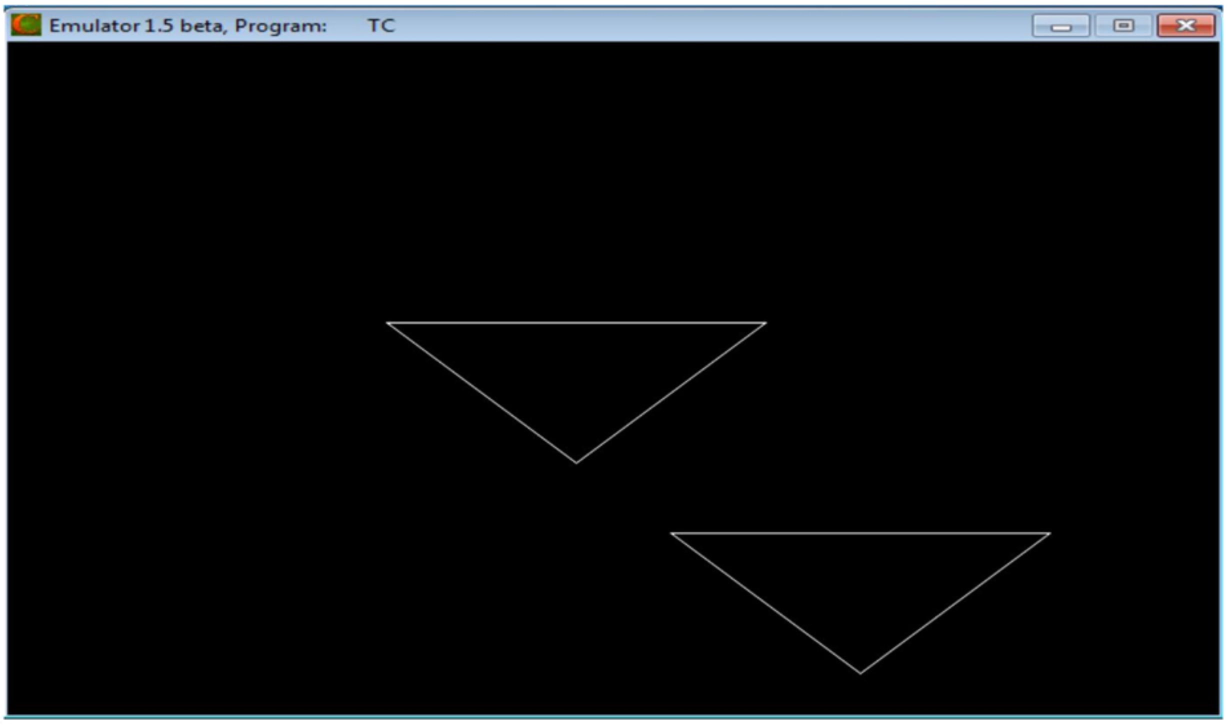
```
                    triangle (x1*x, y1*y, x2*x, y2*y, x3*x, y3*y);
                    closegraph ();
                    break;
                case 4:
                    original ();
                    rotate (180, x2, y2);
                    closegraph ();
                    break;
                case 5:
                    cout<< "\n Enter the Shear Value : \t\t";
                    cin>> x;
                    original ();
                    triangle (x1, y1, x2+x, y2, x3, y3);
                    closegraph ();
                    break;
                case 6:
                    initgraph (&gd, &gm, "C:\\TC\\BGI");
                    rectangle (w1, w2, w3, w4);
                    outtextxy (300, 10, "Window Port");
                    triangle (x1, y1, x2, y2, x3, y3);
                    x = (v3 - v1) / (w3 - w1);
                    y = (v4 - v2) / (w4 - w2);
                    vx1 = v1 + floor (((x1 - w1) * x) + 0.5);
                    vy1 = v2 + floor (((y1 - w2) * y) + 0.5);
                    vx2 = v1 + floor (((x2 - w1) * x) + 0.5);
                    vy2 = v2 + floor (((y2 - w2) * y) + 0.5);
                    vx3 = v1 + floor (((x3 - w1) * x) + 0.5);
                    vy3 = v2 + floor (((y3 - w2) * y) + 0.5);
                    rectangle (v1, v2, v3, v4);
                    outtextxy (450, 85, "View Port");
                    triangle (vx1, vy1, vx2, vy2, vx3, vy3);
                    closegraph ();
                }
        } while (ch<= 6); getch ();
}

void original ( )
{
        initgraph (&gd, &gm, "C:\\TC\\BGI");
        triangle (x1, y1, x2, y2, x3, y3);
}

void triangle (float x1, float y1, float x2, float y2, float x3, float y3)
{
        line (x1, y1, x2, y2);
        line (x2, y2, x3, y3);
        line (x3, y3, x1, y1);
        getch ();
}

void rotate (float t, float rx, float ry)
{
        t = t * (3.14 / 180);
        rx1 = rx + (x1 - rx) * cos (t) - (y1 - ry) * sin (t);
```

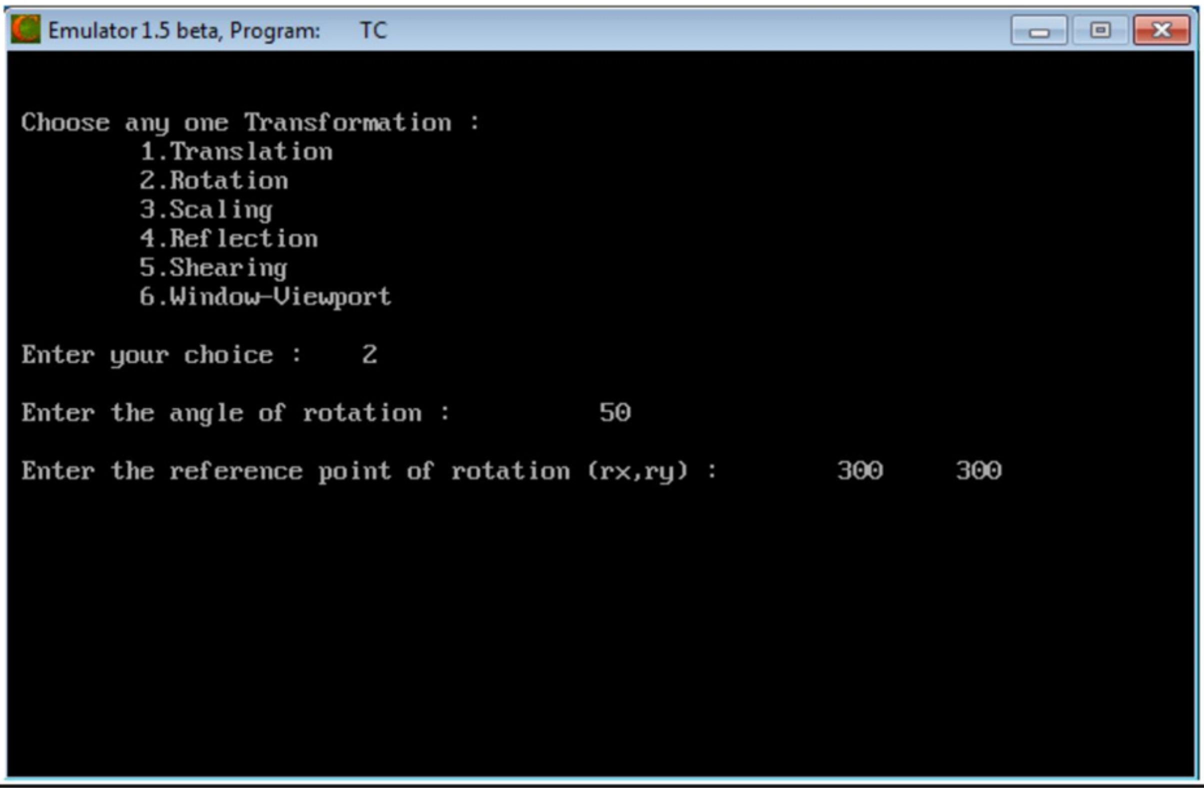
```
            ry1 = ry + (x1 - rx) * sin (t) + (y1 - ry) * cos (t);
            rx2 = rx + (x2 - rx) * cos (t) - (y2 - ry) * sin (t);
            ry2 = ry + (x2 - rx) * sin (t) + (y2 - ry) * cos (t);
            rx3 = rx + (x3 - rx) * cos (t) - (y3 - ry) * sin (t);
            ry3 = ry + (x3 - rx) * sin (t) + (y3 - ry) * cos (t);
            triangle (rx1, ry1, rx2, ry2, rx3, ry3);
    }
```

OUTPUT:

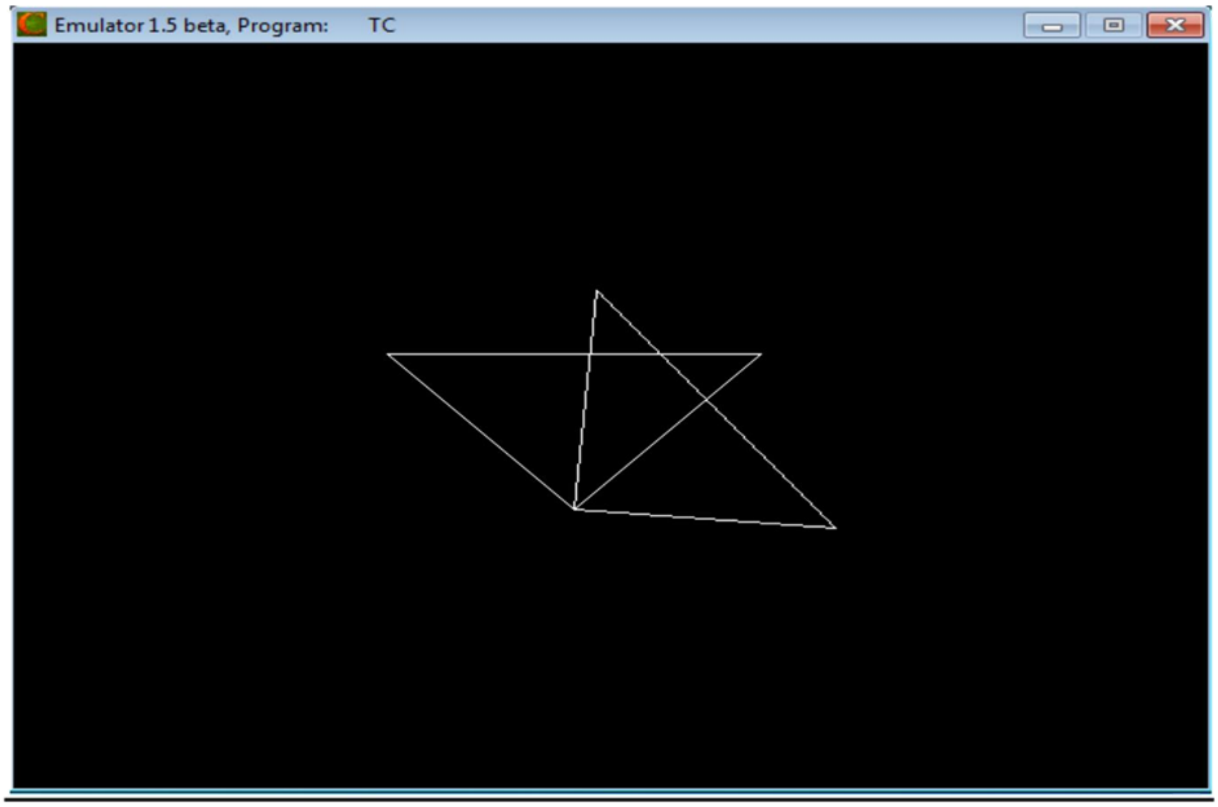


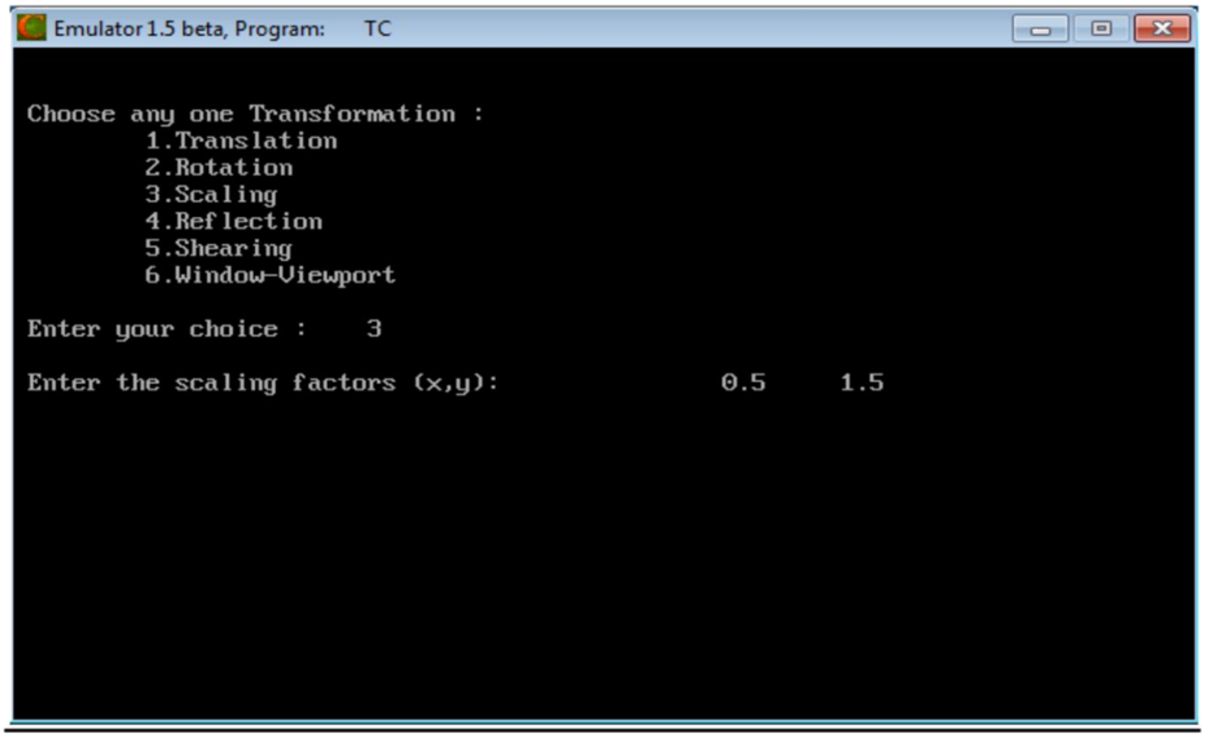
```
                  ***** 2D Geometric Transformations *****

 Enter the coordinates of triangle (x1, y1, x2, y2, x3, y3):
200     200     300     300     400     200
```



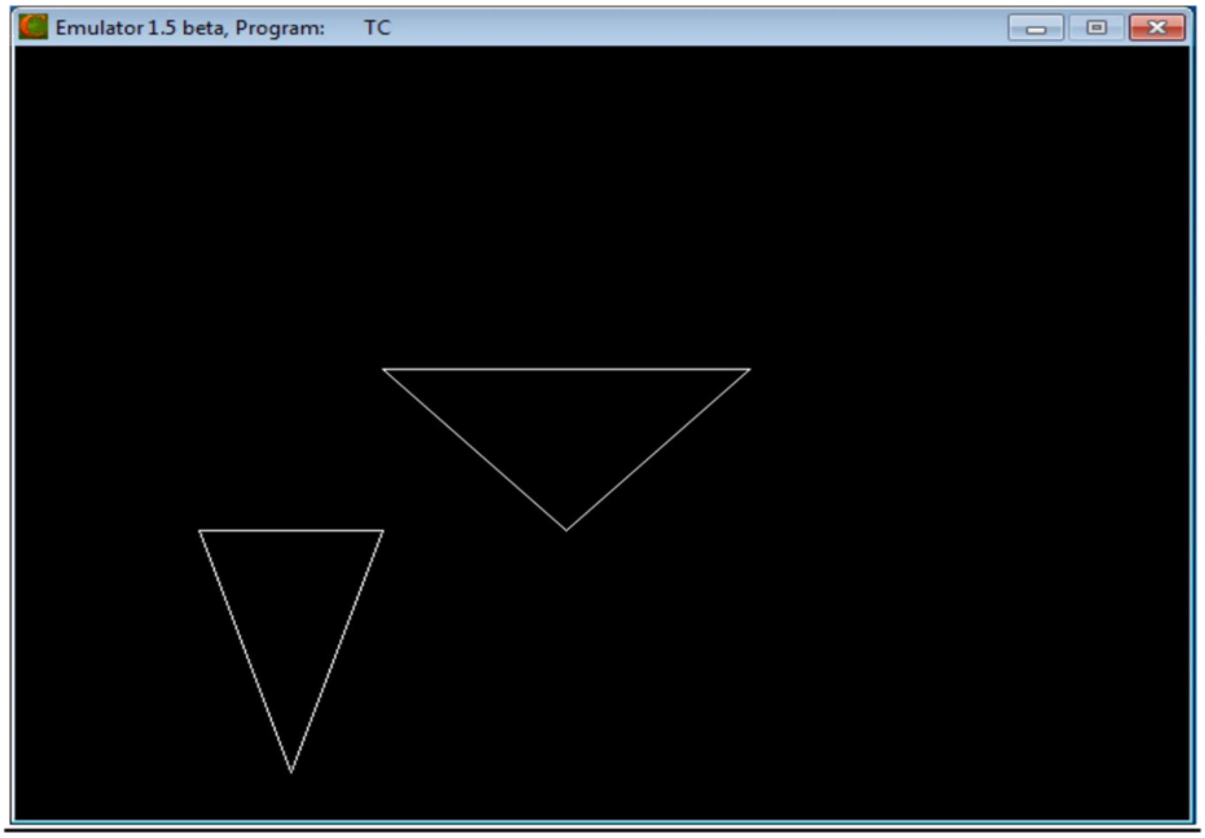


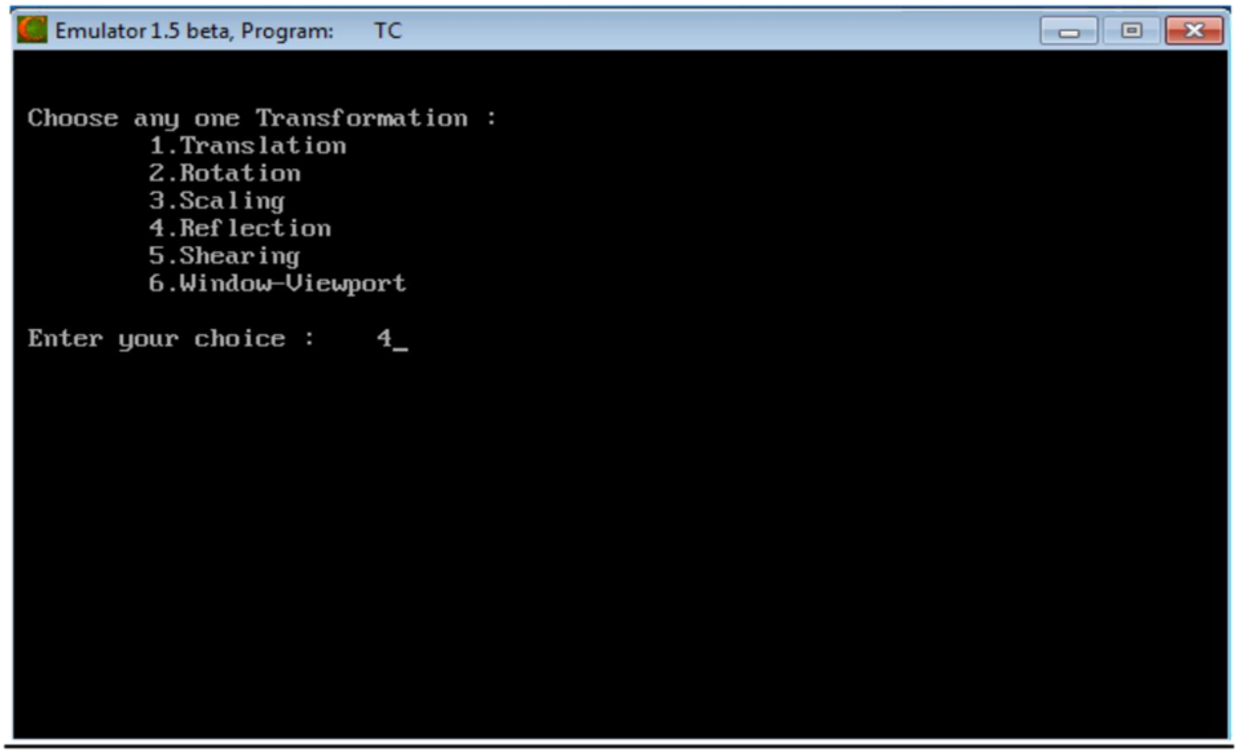
```
Choose any one Transformation :
        1.Translation
        2.Rotation
        3.Scaling
        4.Reflection
        5.Shearing
        6.Window-Viewport

Enter your choice :    2

Enter the angle of rotation :           50

Enter the reference point of rotation (rx,ry) :         300     300
```



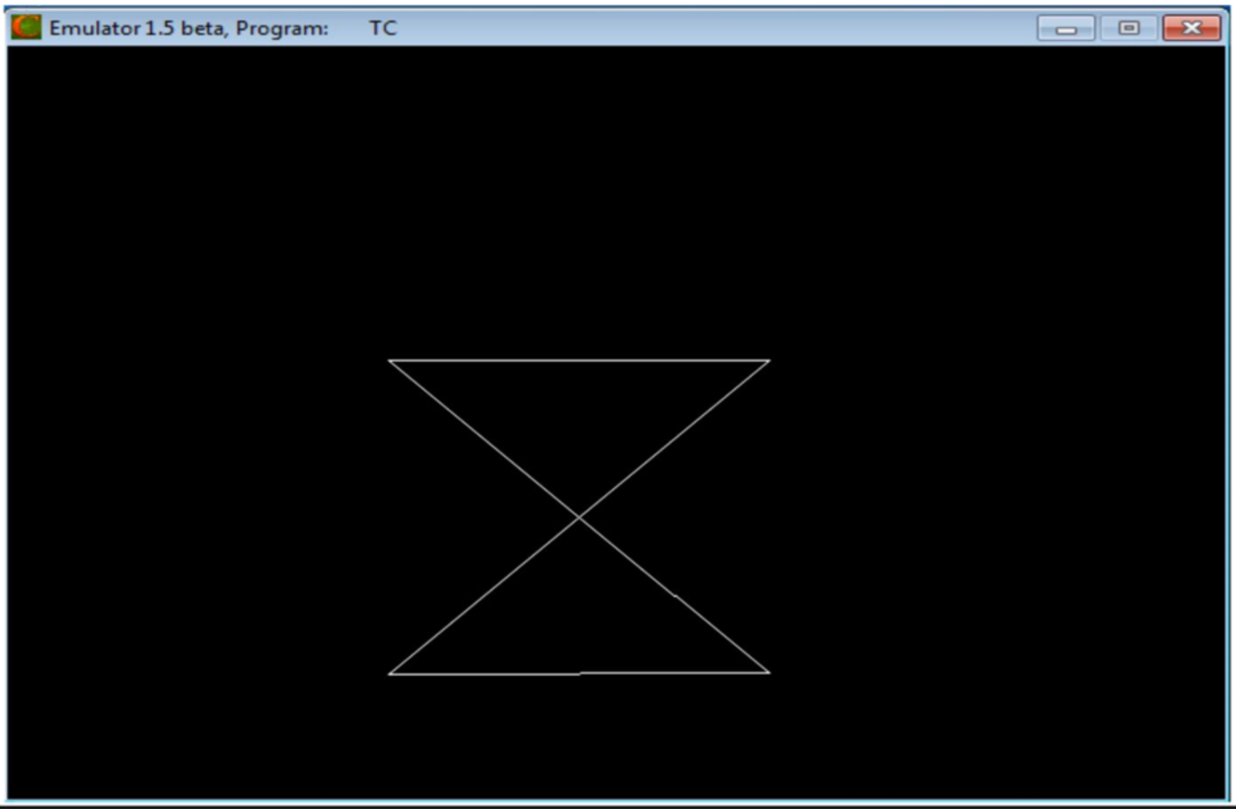


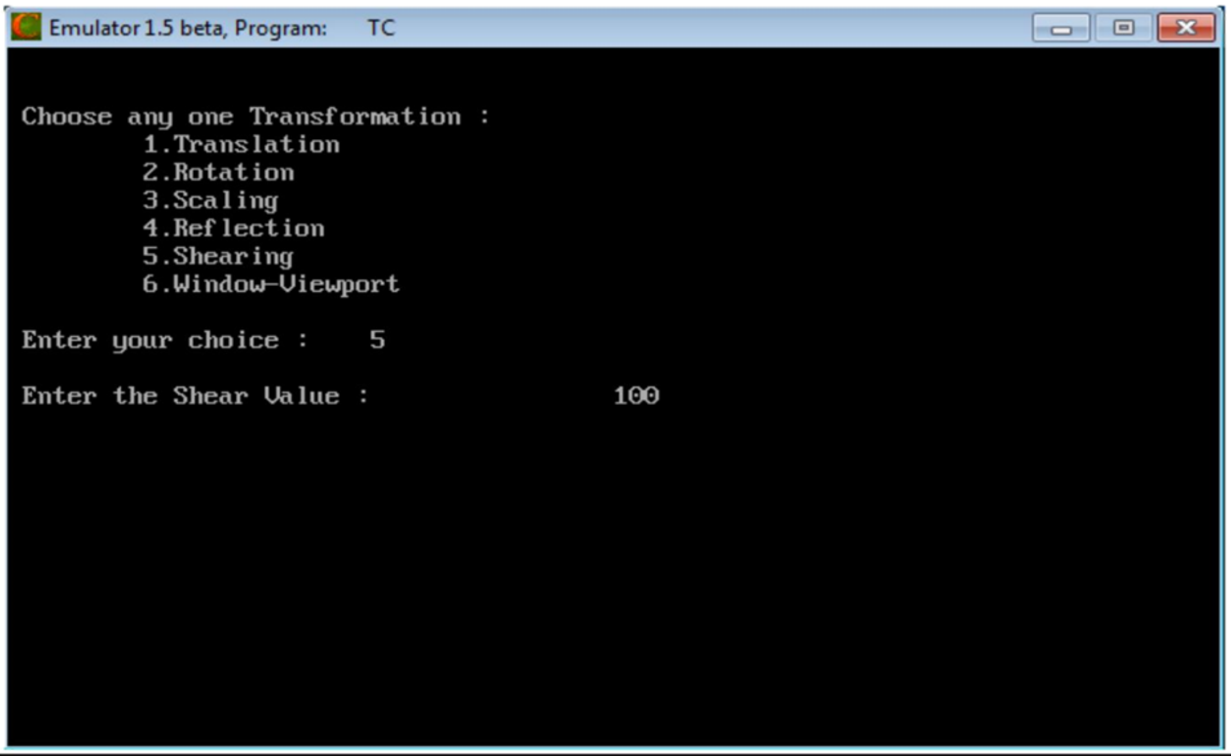
```
Choose any one Transformation :
        1.Translation
        2.Rotation
        3.Scaling
        4.Reflection
        5.Shearing
        6.Window-Viewport

Enter your choice :    3

Enter the scaling factors (x,y):              0.5     1.5
```



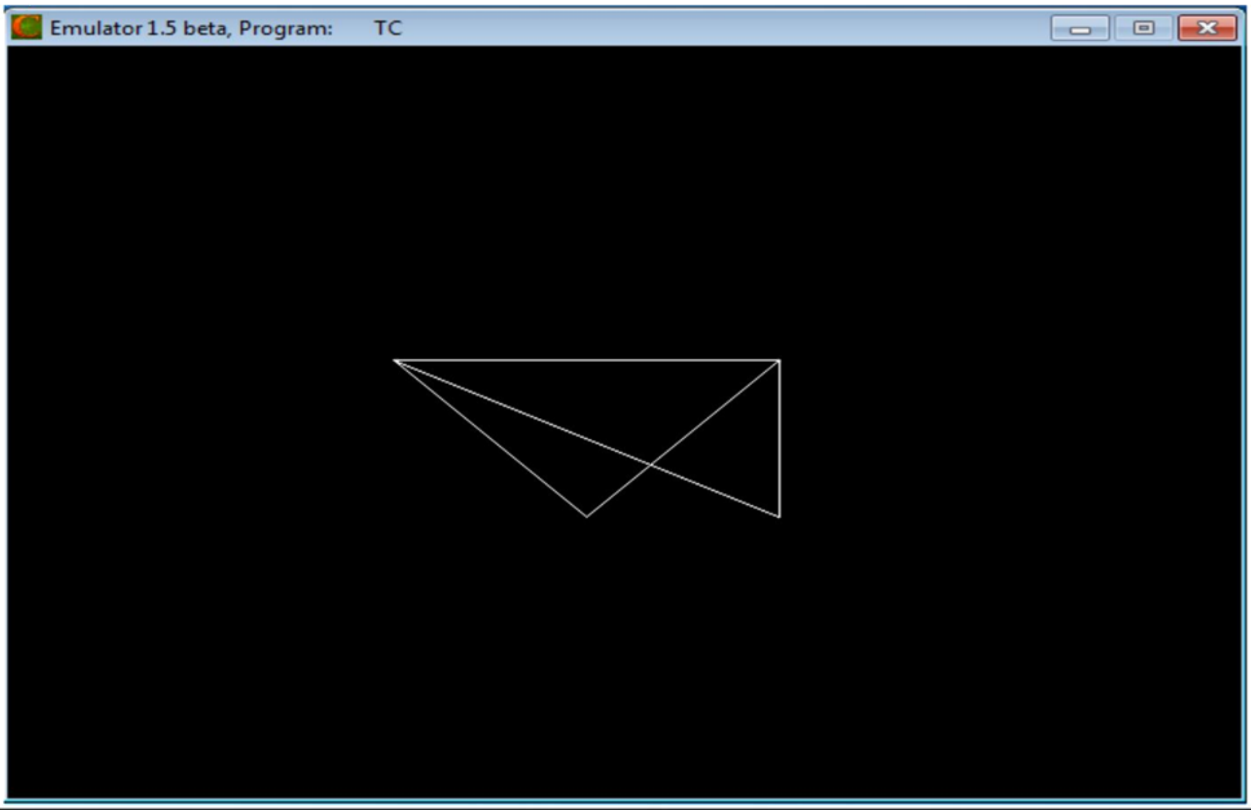


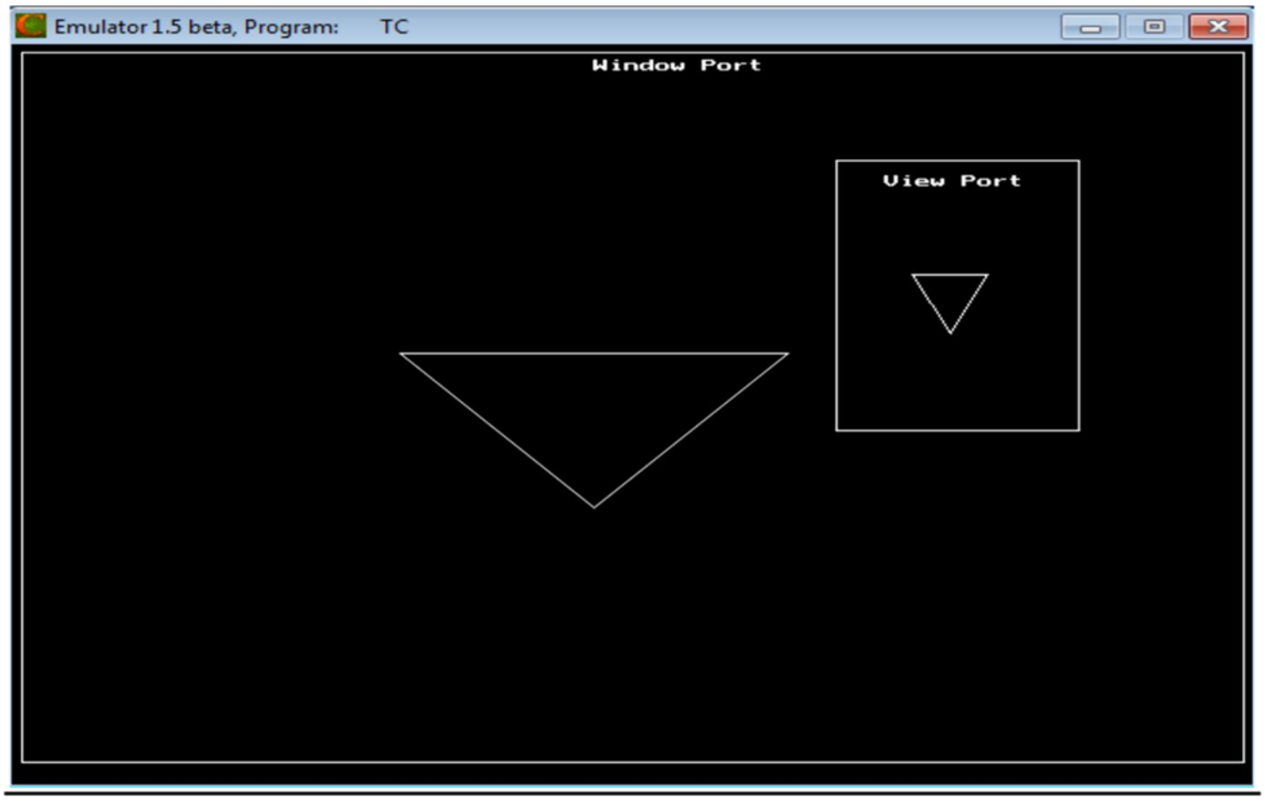
```
Choose any one Transformation :
        1.Translation
        2.Rotation
        3.Scaling
        4.Reflection
        5.Shearing
        6.Window-Viewport

Enter your choice :     4_
```





```
Choose any one Transformation :
        1.Translation
        2.Rotation
        3.Scaling
        4.Reflection
        5.Shearing
        6.Window-Viewport

Enter your choice :    5

Enter the Shear Value :                100
```


Emulator 1.5 beta, Program: TC



RESULT:

Thus the implementation of the 2D Geometric Transformations (translation, rotation, scaling, reflection, shearing) has been implemented and the output was verified.

4. COHEN - SUTHERLAND LINE CLIPPING AlGORITHM

AIM:

To write a program to implement the line clipping.

ALGORITHM:

1. Get the clip window coordinates.
2. Get the line end points.
3. Draw the window and the line.
4. Remove the line points which are plotted in outside the window.
5. Draw the window with clipped line.

THEOREY:

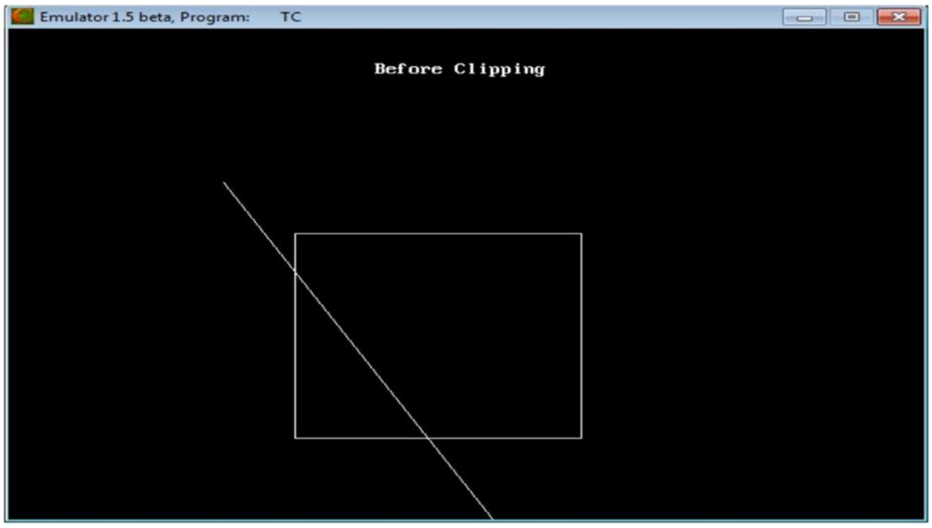
In computer graphics, the Cohen–Sutherland algorithm is an algorithm used for line clipping. The algorithm divides a two-dimensional space into 9 regions and then efficiently determines the lines and portions of lines that are visible in the central region of interest (the viewport).

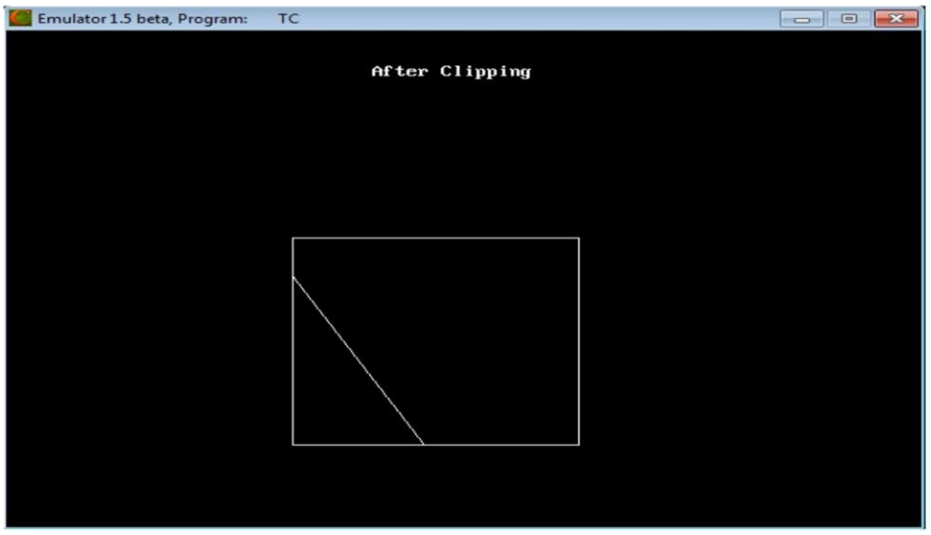
PROGRAM:

```
#include<graphics.h>
#include<iostream.h>
#include<conio.h>
float x1,y1,x2,y2,wx1,wy1,wx2,wy2,m;
intgd,gm;
void main()
{


        clrscr();
        cout<<"\nEnter the clip window coordinates : ";
        cin>>wx1>>wy1>>wx2>>wy2;
        cout<<"\nEnter the line end points : ";
        cin>>x1>>y1>>x2>>y2;
        m=(y2-y1)/(x2-x1);
        initgraph(&gd,&gm,"C:\\TC\\BGI");
        cout<<"\n\n\t\t\t\tBefore Clipping";
        rectangle(wx1,wy1,wx2,wy2);
        line(x1,y1,x2,y2);
        getch();
        closegraph();
        if(x1<wx1)
        {
                y1=y1+(wx1-x1)*m;
                x1=wx1;
        }
        if(x2>wx2)
        {
```

OUTPUT:

```
Enter the clip window coordinates: 200   200   400   400
Enter the line end points: 150   150   350   500
```





RESULT:

Thus the program Cohen - Sutherland line clipping has been implemented and the output was verified

5.3D TRANSFORMSTION - TRANSLATION, ROTATION, SCALING

AIM:

To implement the following 3D Transformations:

a. Translation b. Rotation c. Scaling

ALGORITHM:

1. Assign the value of edge [20] [3] as a coordinates of cube.
2. Print the menu for choosing 3D Transformation.
3. If user choose translation then get the translation factor (tx, ty, tz) and draw the translated cube.
4. If user choose rotation then print the menu for choosing rotation axis and get the rotation angle.
   a) If user choose rotation about X axis then rotate the cube along X axis and draw the rotated cube.
   b) If user choose rotation about Y axis then rotate the cube along Y axis and draw the rotated cube.
   c) If user choose rotation about Z axis then rotate the cube along Z axis and draw the rotated cube.
5. If user choose scaling then get the scaling factor (sx, sy, sz) and draw the scaled cube.

PROGRAM:

```
#include<iostream>
#include<math.h>
#include<GL/glut.h>
usingnamespacestd;

typedeffloat Matrix4 [4][4];

Matrix4 theMatrix;
staticGLfloat 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];
floattx,ty,tz;
floatsx,sy,sz;
float angle;
```

```
intchoice,choiceRot;

voidsetIdentityM(Matrix4 m)
{
for(int i=0;i<4;i++)
for(int j=0;j<4;j++)
m[i][j]=(i==j);
}

voidtranslate(inttx,intty,inttz)
{

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;
}
}
voidscale(intsx,intsy,intsz)
{
theMatrix[0][0]=sx;
theMatrix[1][1]=sy;
theMatrix[2][2]=sz;
}



voidRotateX(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);

}
voidRotateY(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);

}
voidRotateZ(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);

}
voidmultiplyM()
{
//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];
       }
   }
}

}
voidAxes(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();
}
voiddraw(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]);
```

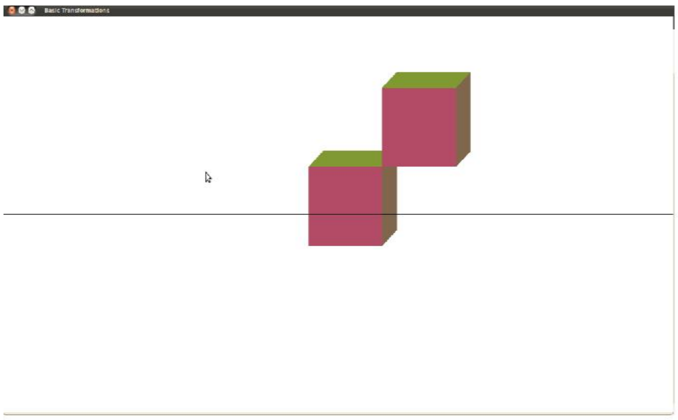
```
glVertex3fv(a[7]);

glEnd();
}

voidinit()
{
glClearColor(1.0,1.0,1.0,1.0); //set backgrondcolor 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
}




voiddisplay()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
Axes();
glColor3f(1.0,0.0,0.0);
draw(input);
setIdentityM(theMatrix);
switch(choice)
{
case1:
translate(tx,ty,tz);
break;
case2:
scale(sx,sy,sz);
multiplyM();
break;
case3:
switch (choiceRot) {
case1:
RotateX(angle);
break;
case2: RotateY(angle);
break;
case3:
RotateZ(angle);
```

```
break;
default:
break;
    }
multiplyM();
break;
}

draw(output);
glFlush();
}

intmain(intargc, 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) {
case1:
cout<<"\nEnterTx,Ty&Tz: \n";
cin>>tx>>ty>>tz;
break;
case2:
cout<<"\nEnterSx,Sy&Sz: \n";
cin>>sx>>sy>>sz;
break;
case3:
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) {
case1:
cout<<"\nENter Rotation angle: ";
cin>>angle;
break;
case2:
```

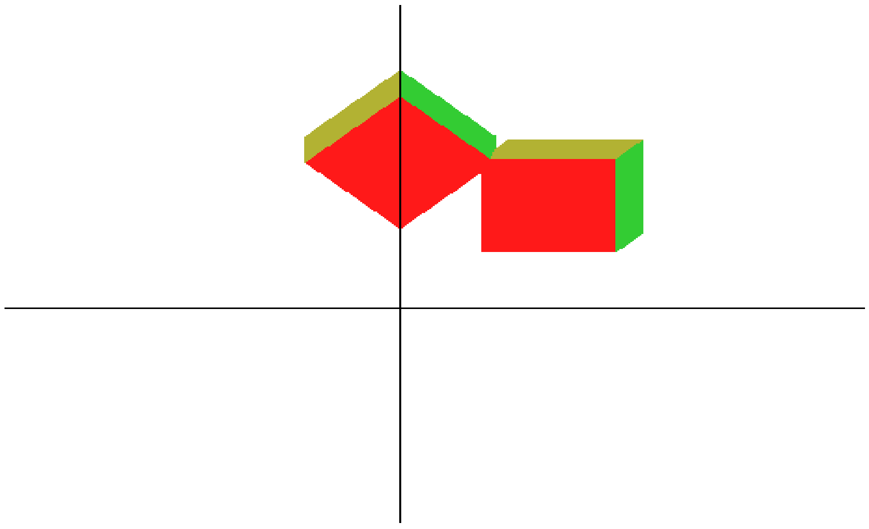
```
cout<<"\nENter Rotation angle: ";
cin>>angle;
break;
case3:
cout<<"\nENter Rotation angle: ";
cin>>angle;
break;
default:
break;
        }
break;
default:
break;
    }
glutDisplayFunc(display);
glutMainLoop();
return0;
}
```

OUTPUT:

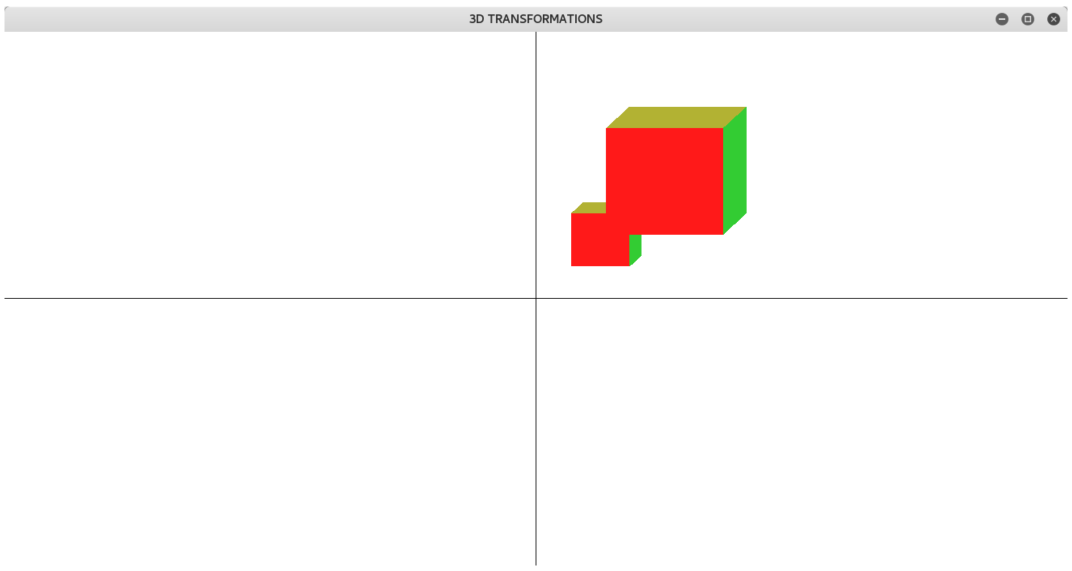
Translation



Rotation



Scaling



RESULT:
Thus the program 3D Transformations - Translation, Rotation,Scalinghas been implemented and the output was verified

6 . 3D PROJECTIONS – PARALLEL, PERSPECTIVE

AIM:

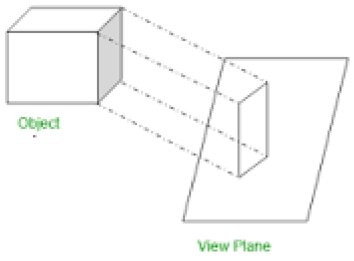
To write a program to implement the following 3D projections:

a. Parallel Projection b. Perspective Projection

ALGORITHM:

```
1. Get the coordinate to draw the cube.
2. Print the menu : 1. Parallel Projection 2. Perspective Projection
3. If user choose Parallel Projection then find the coordinates of following views and draw them:
     a) Side View
     b) Front View
     c) Top View
4.If user choose Perspective Projection then simply draw the cube using bar3d( ) function.
```

THEORY:



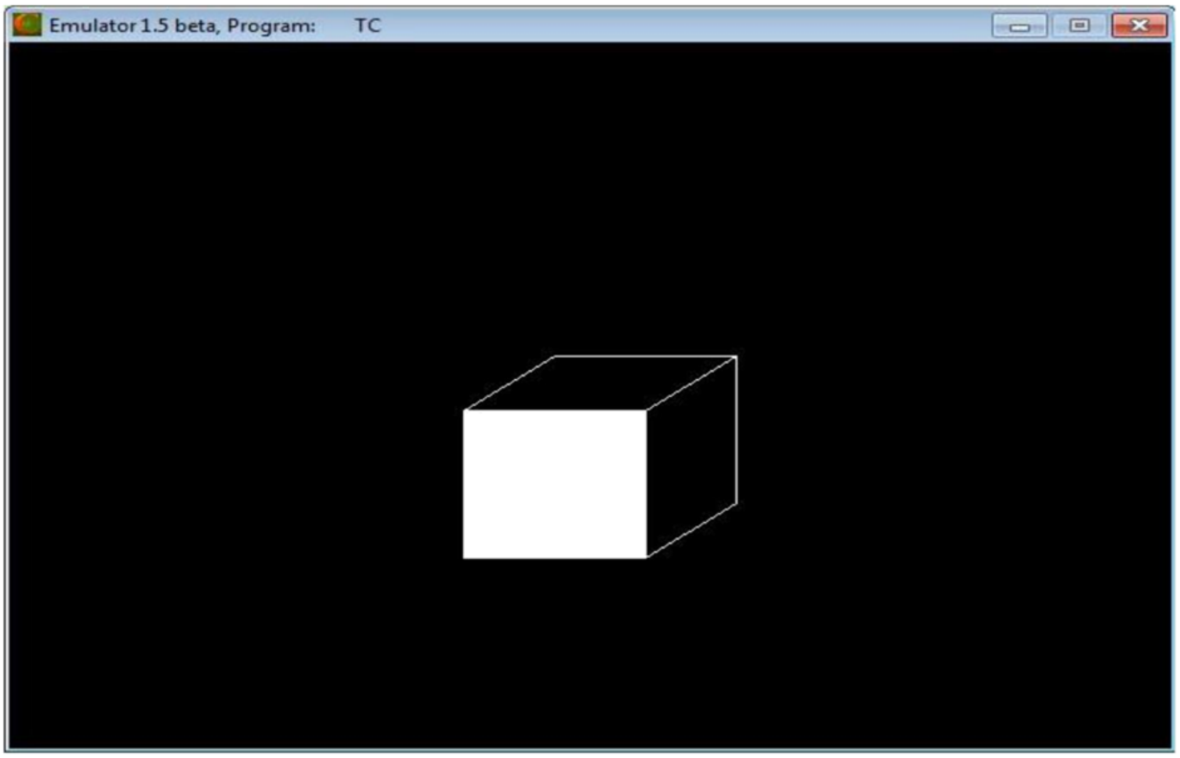
Parallel projection represents the object in a different way like telescope. Perspective projection represents the object in three dimensional way. 2. In parallel projection, these effects are not created. In perspective projection, objects that are far away appear smaller, and objects that are near appear bigger.

PROGRAM:

```
 4.#include<graphics.h>
 #include<iostream.h>
#include<conio.h>
int gd,gm,x1,y1,x2,y2,dep,ch;
void main()
{5.
cout<<"\n Enter the TOP-LEFT and BOTTOM-RIGHT CORNER:";
 6. cin>>x1>>y1>>x2>>y2;
cout<<"\n Enter the depth along z axis:";
cin>>dep;
do
{
        cout<<"Choose any one projection:\n\t1.Parallel Projection\n\t2.Perspective
        Projection\nEnter your choice:";
        cin>>ch;
        initgraph(&gd,&gm,"C:\\TC\\BGI");
        switch(ch)
        {
        case 1:
        rectangle(x2+100,y1,x2+100+dep,y2);
        outtextxy(x2+100,y1-10,"SIDE VIEW");
        rectangle(x1,y1,x2,y2);
        outtextxy(x1,y1-10,"FRONT VIEW");
        rectangle(x1,y1-(y2-y1),x2,x1+dep-(y2-y1));

                        outtextxy(x1,y1-(y2-y1)-10,"TOP VIEW");
                        getch();
                        closegraph();
                        break;
                case 2:
                        bar3d(x1,y1,x2,y2,dep,1);
                        getch();
                        closegraph();
                        break;
            }
        }while(ch<3);
}
```

RESULT:

Thus the implementations of 3D projections (Parallel and Perspective) has been created and the output was verified


Emulator 1.5 beta, Program: TC

7.CREATING 3DSCENES

AIM:

To create the 3D Scene in C++.

ALGORITHM:

1. Set the background color.
2. Find the coordinates to draw a 3D scene.
3. Plot that coordinates using pre-defined functions like line( ), rectangle( ), fillellipse( ).
4. Set the color of the objects by setcolor( ).

THEORY:

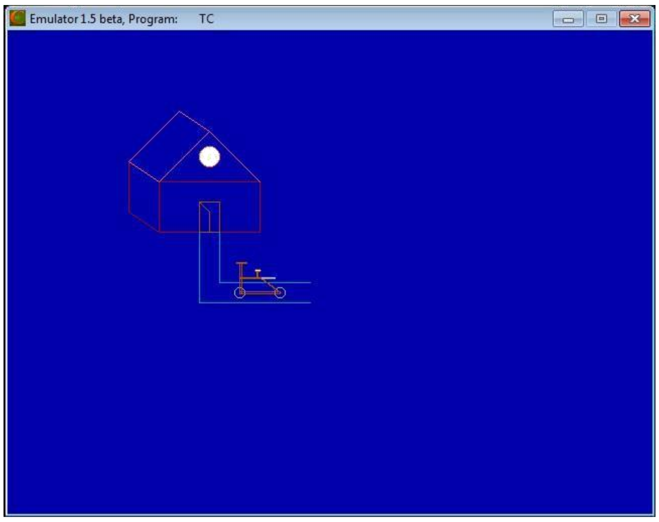
A 3D scene is more than just geometry and a scene without any light would be black, thus a scene also needs lights. In rendering, all this information (the description of the geometry, the camera, and the lights) is contained within a file called the scene file

PROGRAM:

```
#include<graphics.h>
#include<iostream.h>
#include<conio.h>
intgd,gm;
void main()
{
        initgraph(&gd,&gm,"C:\\TC\\BGI");
        setbkcolor(BLUE);
        setcolor(RED);
        rectangle(150,150,250,200);
        line(150,200,120,180);
        line(120,180,120,130);
        setcolor(LIGHTRED);
        line(120,130,150,150);
        line(150,150,200,100);
        line(200,100,250,150);
        fillellipse(200,125,10,10);
        line(120,130,170,80);
        line(170,80,200,100);
        setcolor(BROWN);
        rectangle(190,170,210,200);
        line(190,170,200,180);
        line(200,180,200,200);
        setcolor(CYAN);
        line(190,200,190,270);
        line(210,200,210,250);
        line(190,270,300,270);
        line(210,250,300,250);
```

```
                setcolor(LIGHTGRAY);
                circle(230,260,5);



circle(270,260,5);
            setcolor(BROWN);
            line(230,259,270,259);
            line(230,261,270,261);
            line(230,230,230,261);
            line(232,230,232,260);
            line(227,230,237,230);
            line(227,231,237,231);
            line(270,260,250,245);
            line(271,260,251,245);
            line(250,245,230,245);
            line(250,246,230,246);
            line(248,245,248,240);
            line(247,245,247,240);
            fillellipse(248,238,3,1);
            setcolor(LIGHTGRAY);
            line(252,245,265,245);
            line(252,246,265,246);
            getch();
}
```

OUTPUT:



RESULT:

Thus the 3D scene has been created and the output was verified.

8. COMPRESSION ALGORITHMS – TO IMPLEMENT TEXT AND IMAGE COMPRESSION ALGORITHM.

AIM:

To write a program for the Implementation of Text and Image Compression.

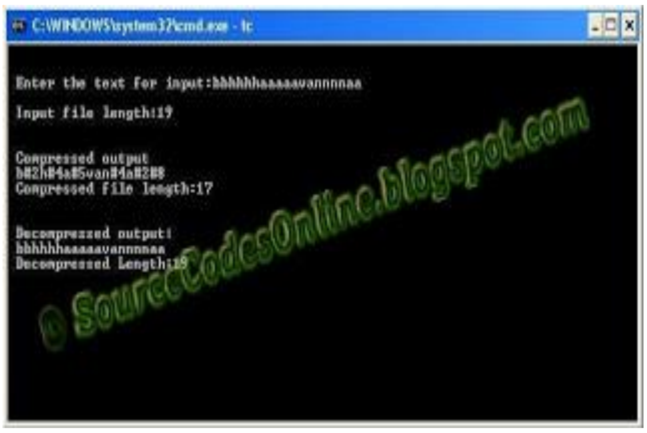
SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
   char a[50],b[50],dl;
   int i,j,flag=0,countindex=0,index=-1,count=0,size;
   FILE *ip,*op;
   clrscr();
   printf("\n\n Enter the text for input:");
   scanf("%s",a);
   size=strlen(a);
   ip=fopen("source","w");
   fprintf(ip,"%s",a);
   printf("\n Input file length:");
   printf("%d",ftell(ip));
   fclose(ip);
   printf("\n");
   ip=fopen("compress.dat","w");
   for(i=0;i<size;i++)
   {
    j=i+1;
    if(a[j]==a[i])
      count=count+1;
    else
      if(count>0)
      {
    b[++index]=a[i];
    fprintf(ip,"%c",b[index]);
    b[++index]='#';
    fprintf(ip,"%c",b[index]);
    b[++index]=count+1;
    fprintf(ip,"%c",b[index]);
    count=0;
      }
    else
```

```
  {
   b[++index]=a[i];
   fprintf(ip,"%c",b[index]);
  }
  countindex=index;
 }
 fclose(ip);
 op=fopen("compress.dat","r+");
 printf("\n\n Compressed output\n ");
 for(i=0;i<size;i++)
 {
  fscanf(op,"%c",b[i]);
  if(b[i]=='#')
  {
   flag=1;
   printf("%c",b[i]);
   continue;
  }
  else if(flag==1)
  {
   printf("%d",b[i]);
   flag=0;
  }
  else
   printf("%c",b[i]);
 }
 printf("\n Compressed file length:");
 printf("%d",ftell(op));
 fclose(op);
 flag=0;
 op=fopen("compress.dat","r+");
 ip=fopen("decompress.dat","w");
 printf("\n\n\n Decompressed output:\n ");
 for(i=0;i<=countindex;i++)
 {
  fscanf(op,"%c",b[i]);
  if(b[i]=='#')
   {
    flag=1;
    continue;
   }
  else if(flag==1)
   {
   for(j=0;j<b[i]-1;j++)
    {
     fprintf(ip,"%c",dl);
     printf("%c",dl);
    }
```

```
      flag=0;
     }
    else if(b[i]!='\0')
     {
      fprintf(ip,"%c",b[i]);
      printf("%c",b[i]);
      dl=b[i];
     }
   }
   printf("\n Decompressed Length:");
   printf("%d",ftell(ip));
   fcloseall();
   getch();
}
```

OUTPUT:



```
Enter the text for input:hhhhhhaaaaavannnnaa

Input file length:19


Compressed output
h#2h#4a#5van#4a#2#8
Compressed file length:17


Decompressed output:
hhhhhhaaaaavannnnaa
Decompressed Length:19
```

Image compression

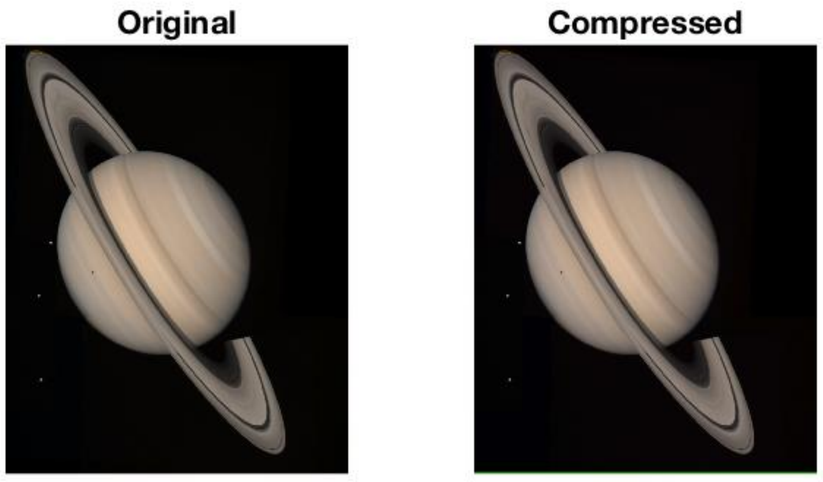
```
length(enco)/ImageSize   % x7 compression.
ans =
   0.147449705335829
```

A different image

```
I = imread('saturn.png');
ImageSize = 8*prod(size(I));
Y_d = rgb2ycbcr( I );
% Downsample:
Y_d(:,:,2) = 2*round(Y_d(:,:,2)/2);
Y_d(:,:,3) = 2*round(Y_d(:,:,3)/2);
% DCT compress:
A = zeros(size(Y_d));
B = A;
for channel = 1:3
    for j = 1:8:size(Y_d,1)-7
        for k = 1:8:size(Y_d,2)-7
            II = Y_d(j:j+7,k:k+7,channel);
            freq = chebfun.dct(chebfun.dct(II).').';
            freq = Q.*round(freq./Q);
            A(j:j+7,k:k+7,channel) = freq;
            % do the inverse at the same time:
            B(j:j+7,k:k+7,channel) = chebfun.idct(chebfun.idct(freq).').';
        end
    end
end
b = A(:);
b = b(:);
b(b==0)=[];  %remove zeros.
b = floor(255*(b-min(b))/(max(b)-min(b)));
symbols = unique(b);
prob = histcounts(b,length(symbols))/length(b);
dict = huffmandict(symbols, prob);
enco = huffmanenco(b, dict);
FinalCompressedImage = length(enco);

FinalCompressedImage/ImageSize
ans =
   0.031958287037037
```

Here's what the images look like:

```
subplot(1,2,1)
imshow(I)
title('Original')
subplot(1,2,2)
imshow(ycbcr2rgb(uint8(B)));
title('Compressed')
```



RESULT:

Thus the To implementation of text and image compression algorithms has been created and the output was verified.

9. IMAGE EDITING AND MANIPULATION - BASIC OPERATIONS ON IMAGE USING ANY IMAGE EDITING SOFTWARE, CREATING GIF ANIMATED IMAGES, IMAGE OPTIMIZATION.

AIM:

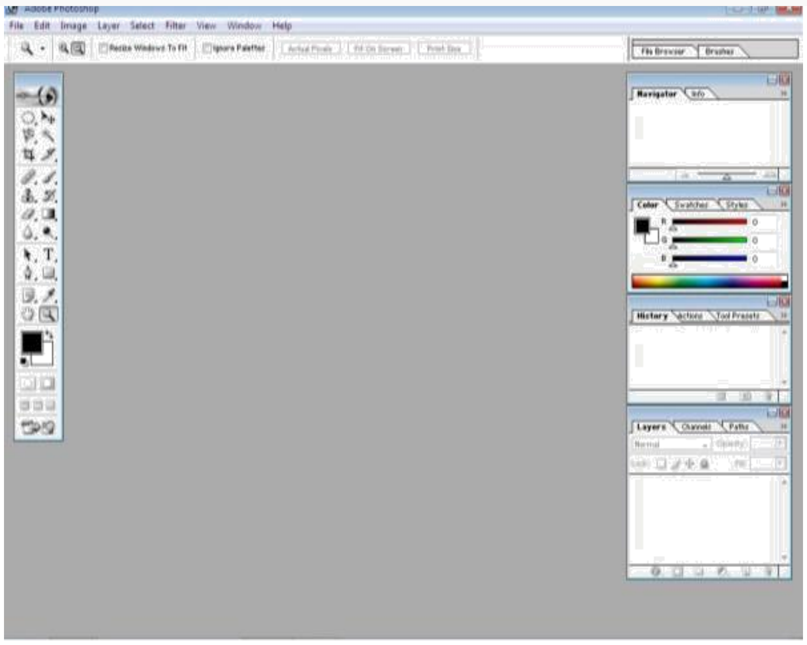
To perform the following operations,

(i) Basic Operations on image using Adobe Photoshop 7.0
(ii) Creating gif animated images in PhotoScape
(iii) To optimize the image in Adobe Photoshop 7.0

PROCEDURE:

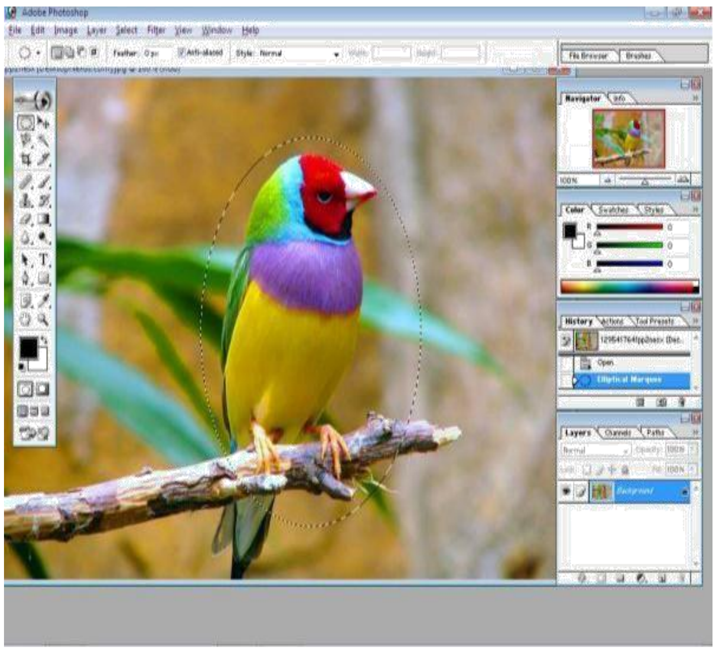
1. **To perform the basic operations on image using Adobe Photoshop 7.0**

   i. Open the Adobe Photoshop 7.0.

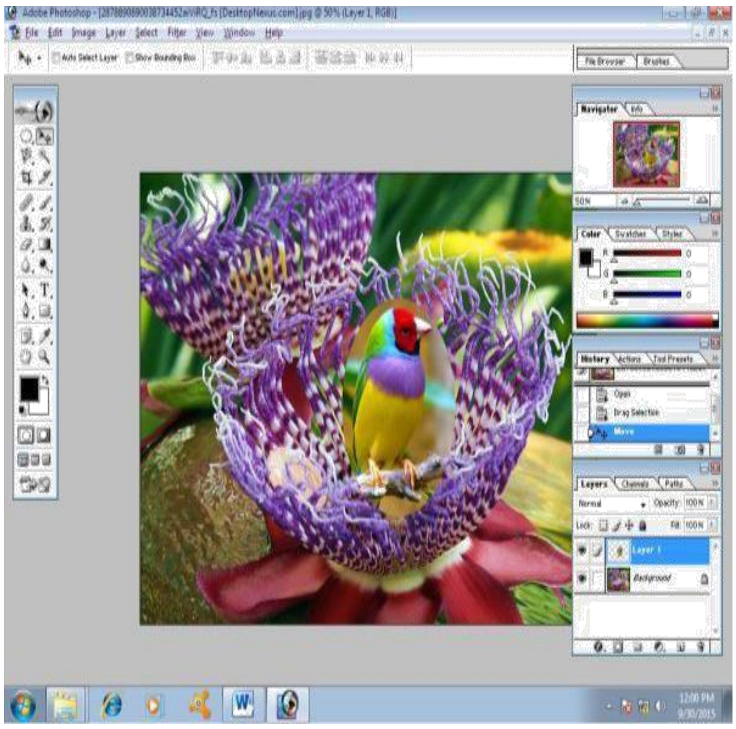


ii. Open the images.

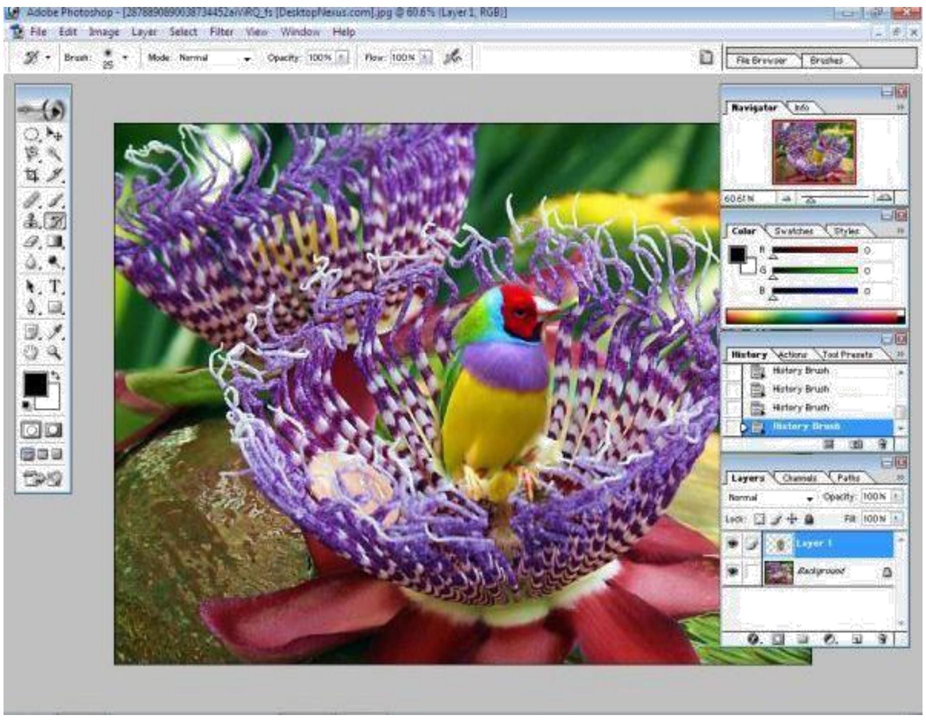
iii) Select the particular portion of the image by Elliptical Marquee Tool



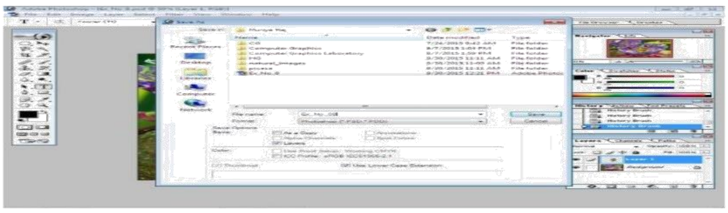
iv. Move that selected portion to another image by using Move Tool.



v . Eliminate the unwanted portion of the image by using History Brush Tool.


Adobe Photoshop - [2878890890038734452aiViRQ_fs [DesktopNexus.com].jpg @ 60.6% (Layer 1, RGB)]
File Edit Image Layer Select Filter View Window Help
Brush: Mode: Normal Opacity: 100% Flow: 100%
File Browser Brushes
Navigator Info
Color Swatches Styles
History Actions Tool Presets
History Brush
History Brush
History Brush
History Brush
Layers Channels Paths
Normal Opacity: 100%
Lock: Fill: 100%
Layer 1
Background

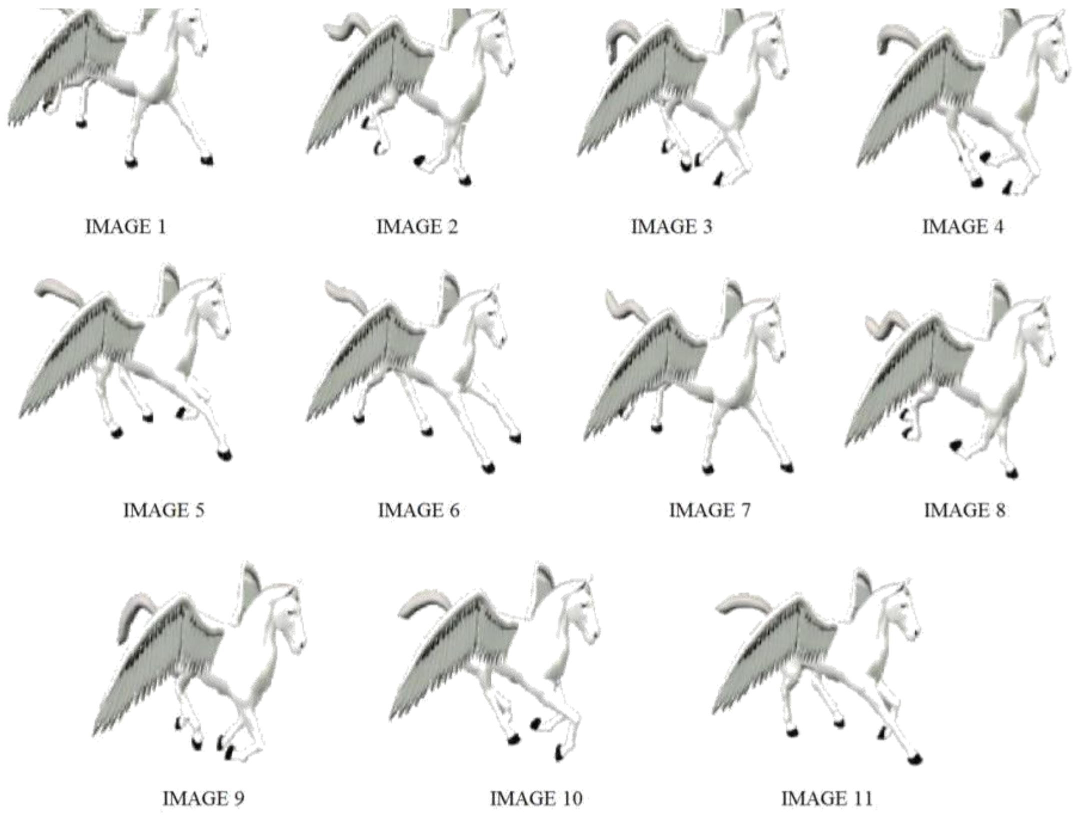
vi. Save the modified image in jpg format



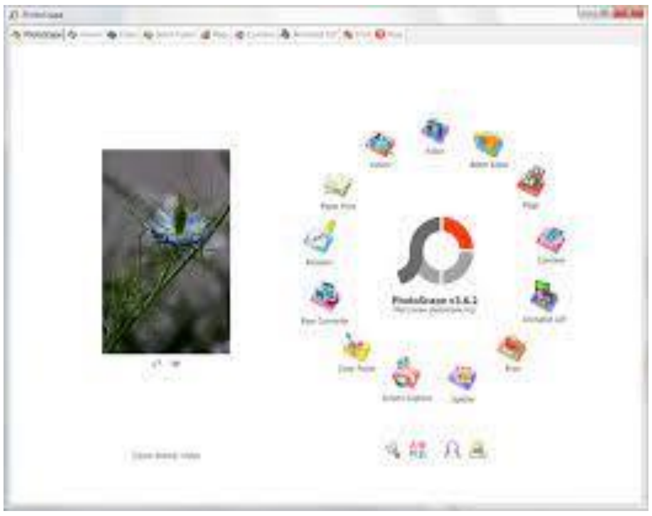
2. **To Create the gif animated images in PhotoScape**

   i. Draw the images in different scenes to animate. Save all the images in jpg format.

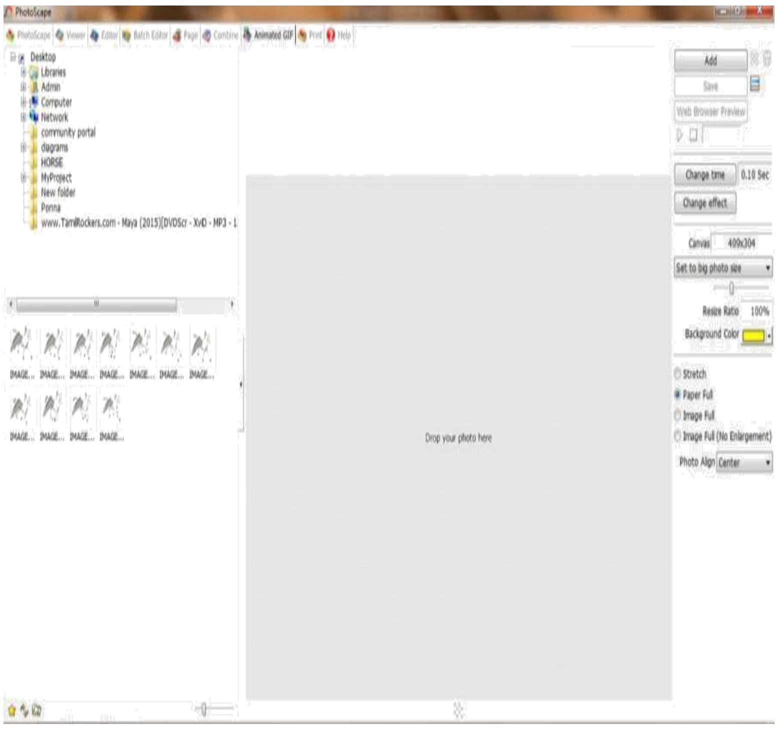
.



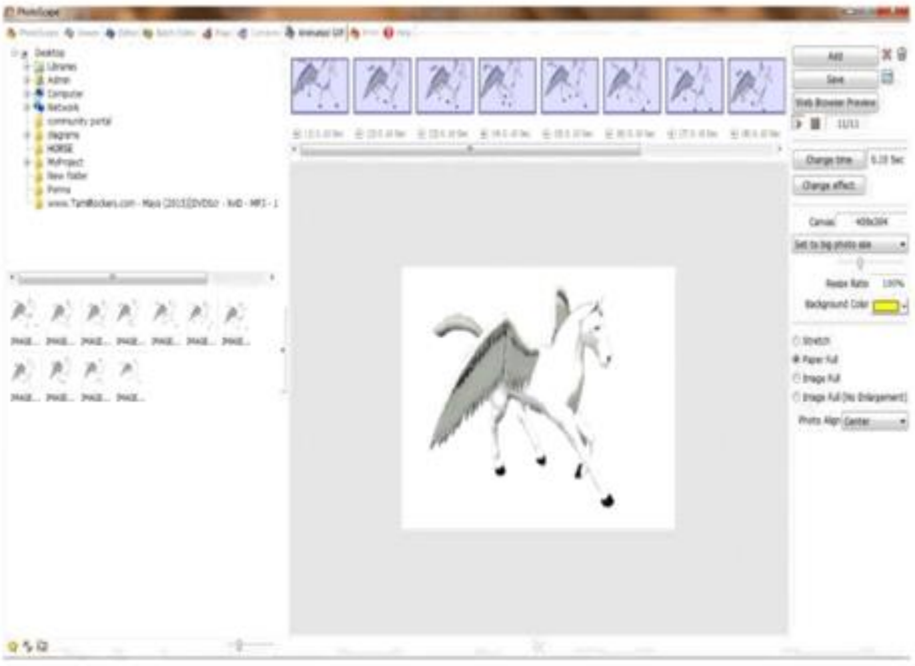
Iii .Open PhotoScape



iii. Choose Animated GIF from the menu.

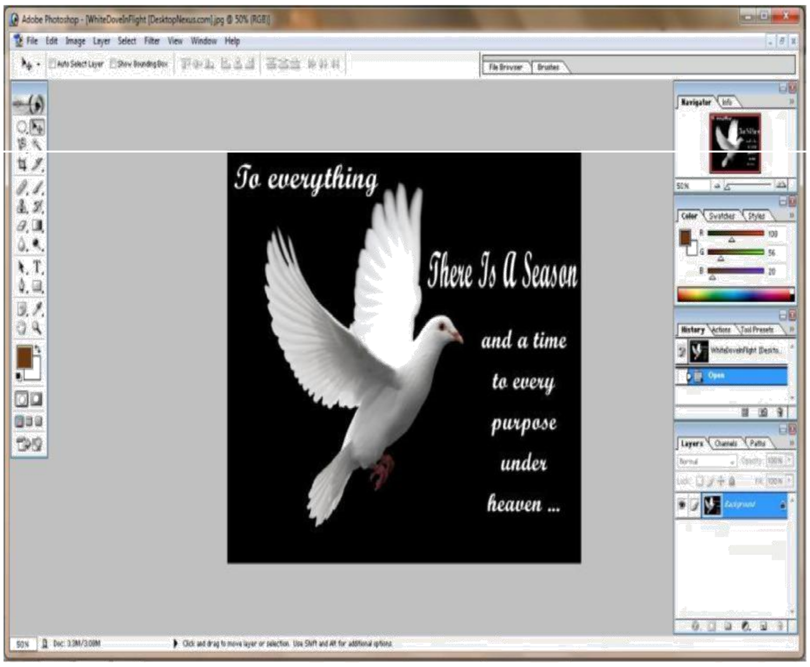


iv. Drop all the images in center bottom portion of the window



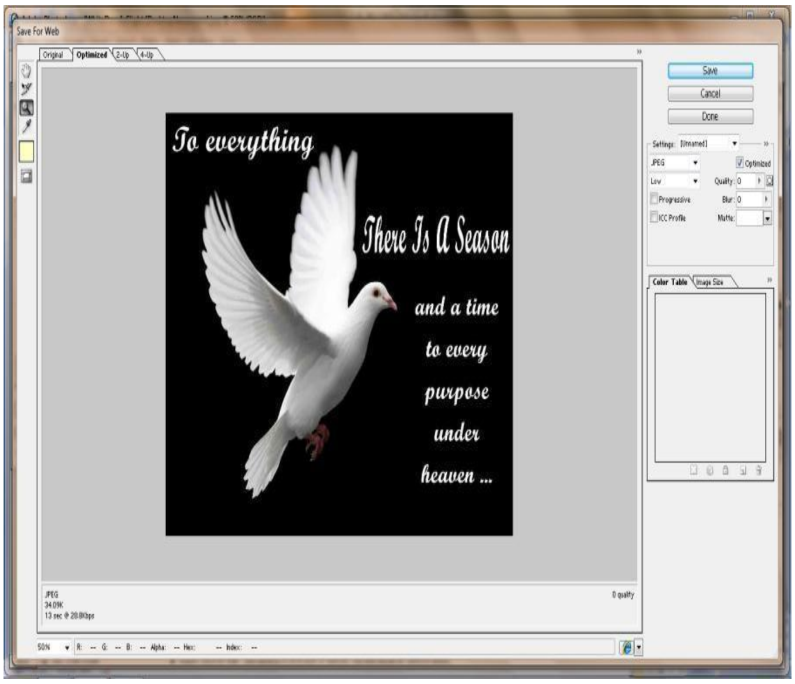
3. **To optimize the image in Adobe Photoshop 7.0**

i.Open the image in Adobe Photoshop 7.0 which size is 85.2 KB

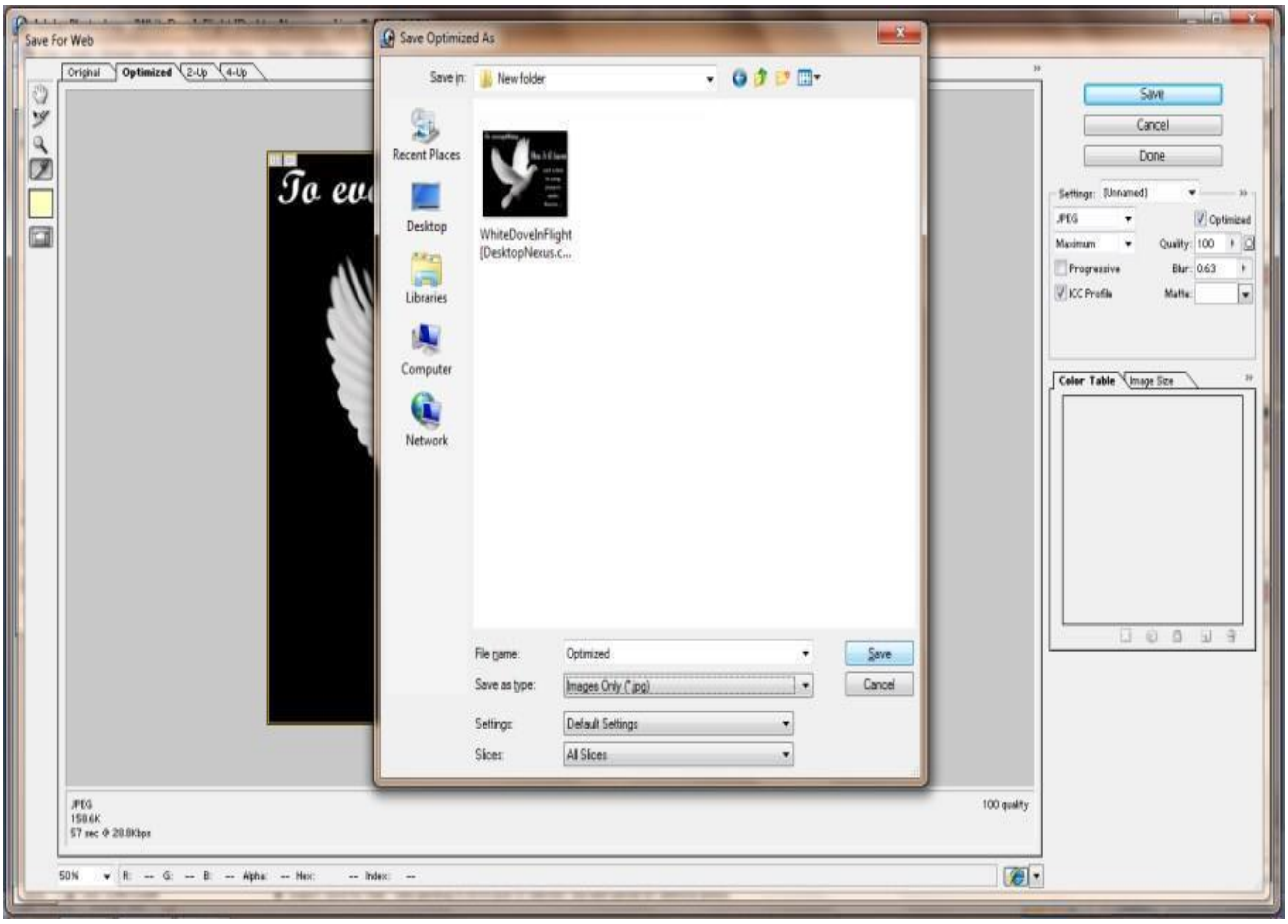

Adobe Photoshop - [WhiteDoveInFlight [DesktopNexus.com].jpg @ 50% (RGB)]
File Edit Image Layer Select Filter View Window Help
To everything
There Is A Season
and a time
to every
purpose
under
heaven ...
Navigator Info
Color Swatches Styles
History Actions Tool Presets
Open
Layers Channels Paths
Background
50%
Doc: 3.3M/3.00M
Click and drag to move layer or selection. Use Shift and Alt for additional options.

•

ii. File   Save for Web (or press Alt + Shift + Ctrl + S)



iii. Adjust the Quality and Blur of the image. Finally the optimized image in jpg format which size is 40.3 KB.



File name: Optimized
Save as type: Images Only (*.jpg)
Settings: Default Settings
Slices: All Slices

Settings: [Unnamed]
JPEG
Maximum
Optimized ✓
Quality: 100
Progressive
Blur: 0.63
ICC Profile ✓
Matte:

JPEG
158.6K
57 sec @ 28.8Kbps
100 quality

RESULT:
Thus the operations (Basic Operations on image, creating gif animated image, optimize the image) has been performed and the output was verified

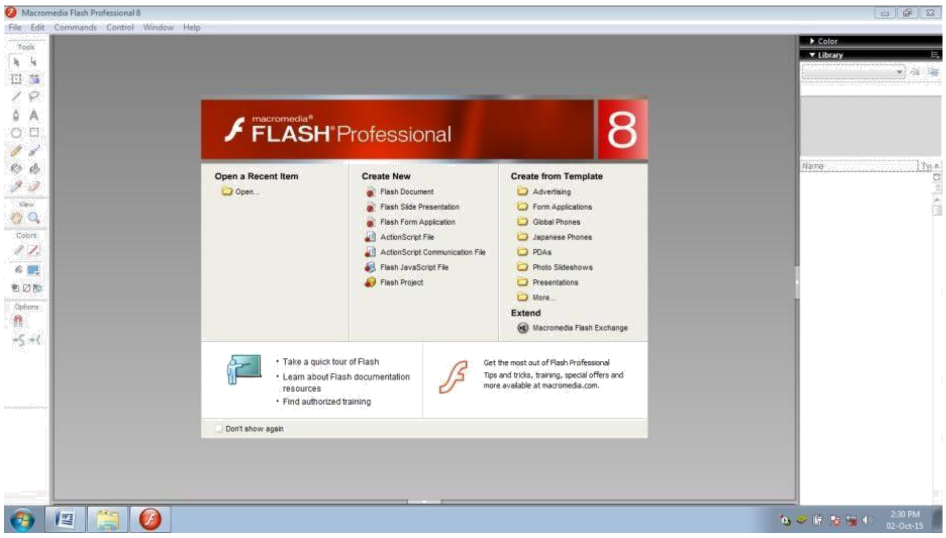
10. 2D ANIMATION – TO CREATE INTERACTIVE ANIMATION USING ANY AUTHORING TOOL

AIM:

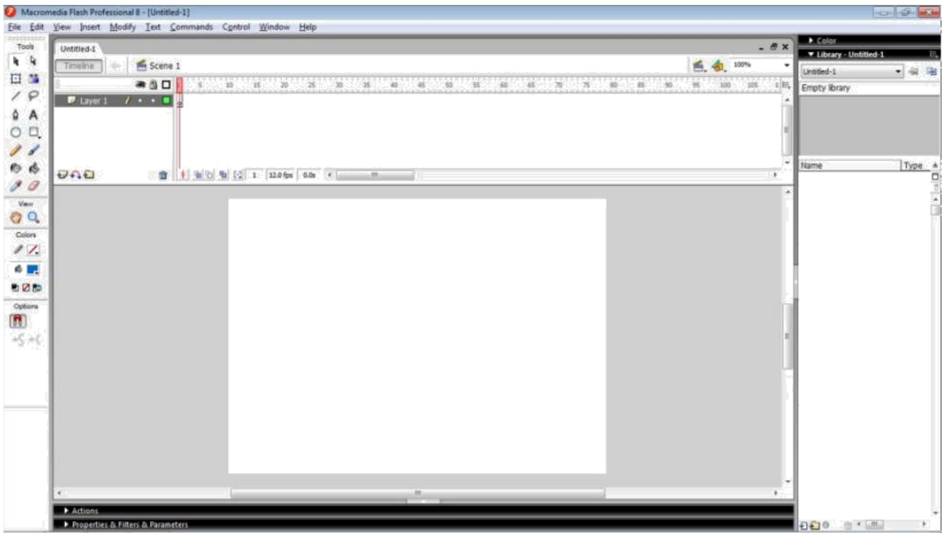
To create interactive animation using Macromedia Flash 8.

PROCEDURE:

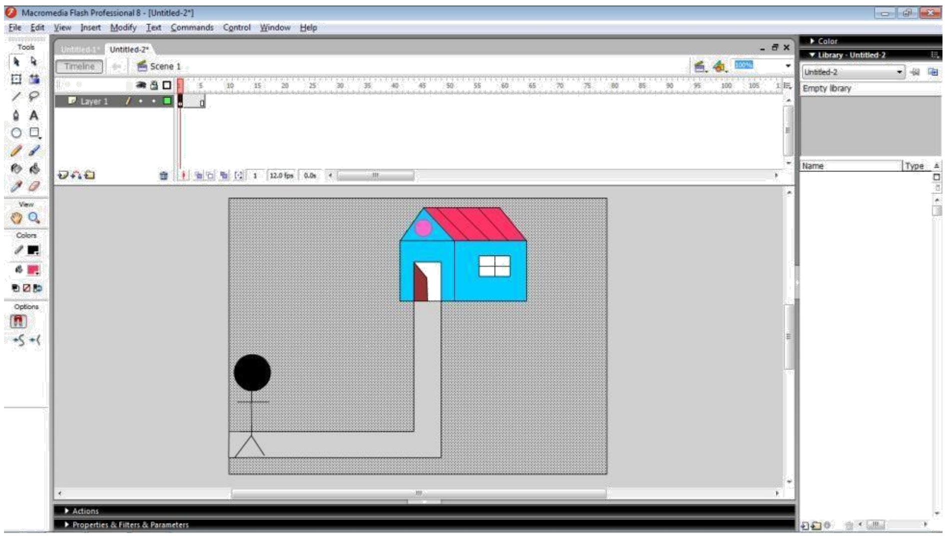
1. Open Macromedia Flash 8.



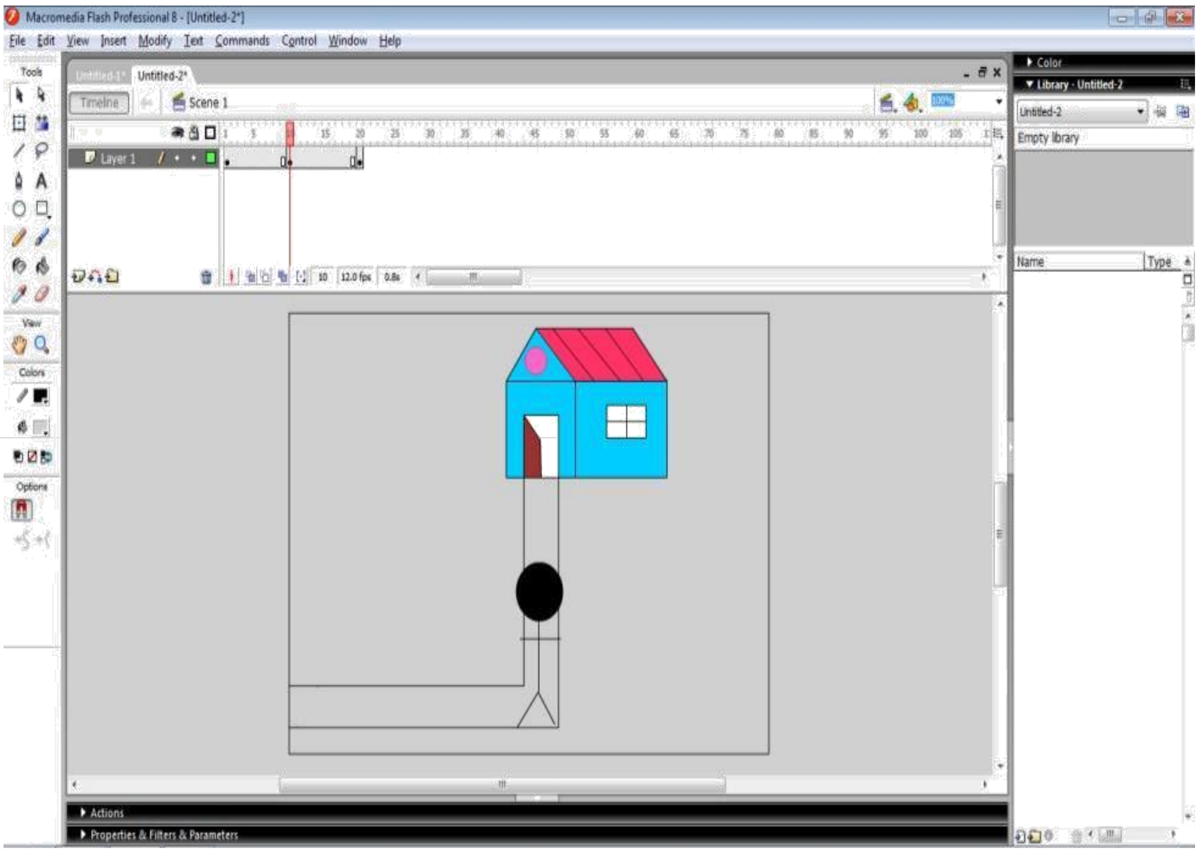
2. Create New • Flash Document.


Macromedia Flash Professional 8 - [Untitled-1]
File Edit View Insert Modify Text Commands Control Window Help
Untitled-1
Timeline
Scene 1
Layer 1
Library - Untitled-1
Empty library
Actions
Properties & Filters & Parameters

Draw the 2D image (scene) to animate.

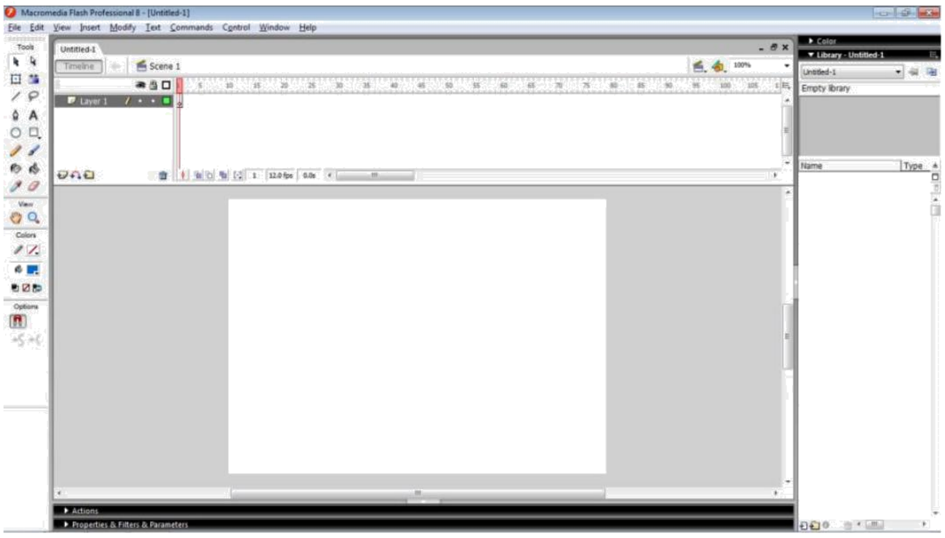

Macromedia Flash Professional 8 - [Untitled-2*]
File Edit View Insert Modify Text Commands Control Window Help
Untitled-2*
Timeline
Scene 1
Layer 1
Color
Library - Untitled-2
Empty library
Actions
Properties & Filters & Parameters

4. Insert a new frame and change the scene for new frame.


Macromedia Flash Professional 8 - [Untitled-2*]
File Edit View Insert Modify Text Commands Control Window Help
Untitled-2*
Timeline
Scene 1
Layer 1
Color
Library - Untitled-2
Empty library
Actions
Properties & Filters & Parameters

5. Similarly create many frames.

•


Macromedia Flash Professional 8 - [Untitled-1]
File Edit View Insert Modify Text Commands Control Window Help
Tools
Untitled-1
Timeline
Scene 1
Layer 1
View
Colors
Options
Color
Library - Untitled-1
Untitled-1
Empty library
Name
Type
12.0 fps
0.0s
Actions
Properties & Filters & Parameters

6. Control Play (or press Enter) to run the Animation.

RESULT:

Thus the interactive 2D Animation has been created and the output was verified.