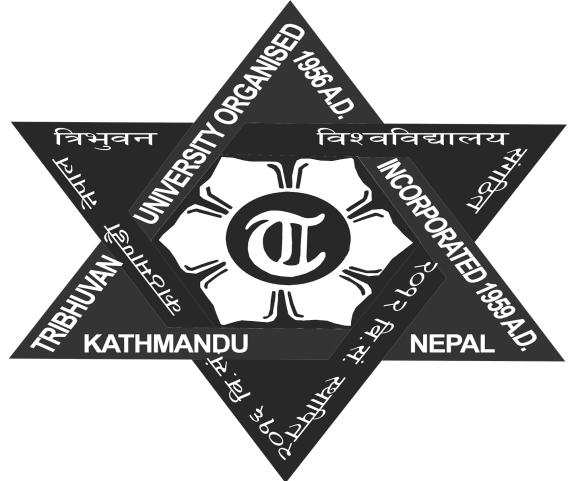
**TRIBHUVAN UNIVERSITY**

**Faculty of Humanities and Social Science.**

**Bajra International College**

**Boudha, Jorpati, Kathmandu**



**Lab Report**

**Submitted To**:

Department of BCA

Bajra International College

**Submitted By**:

Name: Milan Karki

Roll No: 05

Class: BCA 5th Semester

Subject: Computer Graphics and Animation (CACS 305)

**Contents**

[LAB 1 WAP TO DRAW LINE 1](#_Toc114428419)

[LAB 2 WAP TO DRAW CIRCLE 2](#_Toc114428420)

[LAB 3 WAP TO DRAW HOUSE 3](#_Toc114428421)

[LAB 4.1 WAP TO DRAW FLOOR FILLING 5](#_Toc114428422)

[LAB 4.2 WAP TO DRAW BOUNDARY FILLING 8](#_Toc114428423)

[LAB 5.1 WAP TO DRAW LINE USING BRESENHAM ALGORITHM 11](#_Toc114428424)

[LAB 5.2 WAP TO DRAW LINE USING DDA. 14](#_Toc114428425)

[LAB 6 WAP TO DRAW USING MID POINT CIRCLE ALGORITHM. 17](#_Toc114428426)

[LAB 7 WAP TO DRAW USING MID POINT ELLIPSE ALGORITHM. 20](#_Toc114428427)

[LAB 8 WAP TO IMPLEMENT 2D TRANSFORMATION 23](#_Toc114428428)

[LAB 9 WAP TO IMPLEMENT 3D TRANSFORMATION 27](#_Toc114428429)

[LAB 10 WAP TO IMPLEMENT SUNDERLAND LINE CLIPPING 32](#_Toc114428430)

[LAB 11 WAP TO SHOW BOUNCING BALL 39](#_Toc114428431)

[LAB 12 INTRODUCTION TO FLASH 41](#_Toc114428432)

**Acknowledgement**

This project has been prepared for the partial fulfillment of the requirement for BCA Fifth Semester Computer Graphics and Animation course designed by TU.

Knowledge is not just limited on our books and our words; it varies on our experience, on the way facing the time and situation that passes across us. The project work on Computer Graphics and Animation is an excellent way to collaborate the knowledge in our mental attitudes in an IT sector.

The project is a successful work, and this project is a perfect symbolization of knowledge, friends and teacher. First of all, I would like to thank my parents who help me a lot by providing suitable environment more or less, accessories and economical support required for the project. Again, I would like to express my gratitude and appreciation to all who contributed directly or indirectly while preparing this project.

In this project given by our Computer Graphics and Animation Teacher Mr. Saroj Ghimire of Bajra International College I have investigated and applied the use of Adobe Flash Interface and C. The main aim of making this project is to know about Adobe Flash Interface and its function. By doing this project was able to understand about different uses and application of software and present it as example through my project.

Really, this project is an excellent example of co-ordinate and united team as well as other helpful faces and hands.

**Thank You !!!**

**Roll No: 5**

**Name: Milan Karki**

**Semester: BCA Fifth Sem**

**Level: Bachelors in Computer Applications**

# 

# LAB 1 WAP TO DRAW LINE

**THEORY:**

**graphics.h** library is used to include and facilitate graphical operations in program. graphics.h functions can be used to draw different shapes, display text in different fonts, change colors and many more. Using functions of graphics.h we can make graphics programs, animations, projects and games. We can draw circles, lines, rectangles, bars and many other geometrical figures.we can change their colors using the available functions and fill them.

**SOURCE CODE:**

#include <graphics.h>

#include <conio.h>

int main(){

int gd = DETECT, gm;

initgraph(&gd, &gm, "C:/TURBOC3/BGI");

line(100,100,200,100); //will draw a horizontal line

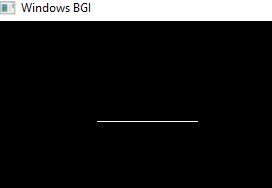
getch();

closegraph();

return 0;

}

**OUTPUT:**

****

# LAB 2 WAP TO DRAW CIRCLE

**THEORY:**

**graphics.h** library is used to include and facilitate graphical operations in program. graphics.h functions can be used to draw different shapes, display text in different fonts, change colors and many more. Using functions of graphics.h we can make graphics programs, animations, projects and games. We can draw circles, lines, rectangles, bars and many other geometrical figures.we can change their colors using the available functions and fill them.

**SOURCE CODE:**

#include<graphics.h>

int main(){

int gd=DETECT, gm;

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

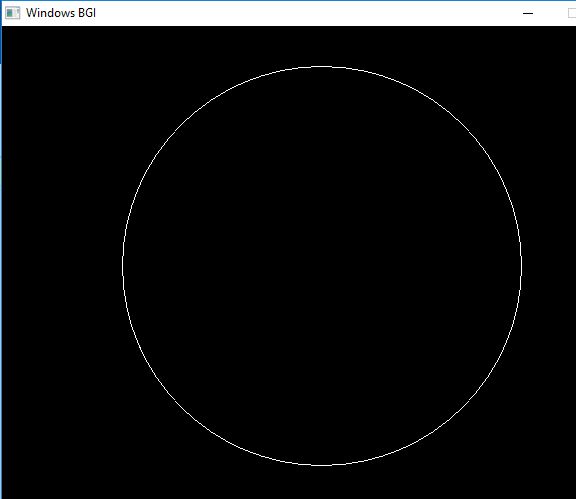
circle(320,240,200);

getch();

closegraph();

return 0;

}

**OUTPUT:**

# LAB 3 WAP TO DRAW HOUSE

**THEORY:**

**graphics.h** library is used to include and facilitate graphical operations in program. graphics.h functions can be used to draw different shapes, display text in different fonts, change colors and many more. Using functions of graphics.h we can make graphics programs, animations, projects and games. We can draw circles, lines, rectangles, bars and many other geometrical figures.we can change their colors using the available functions and fill them.

**SOURCE CODE:**

#include<graphics.h>

int main(){

int gd= DETECT, gm;

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

line(200,100,400,100);

line(200,100,100,200);

line(100,200,210,200);

line(400,100,500,200);

line(500,200,390,200);

line(390,200,390,400);

line(210,200,210,400);

line(210,400,390,400);

/\*DOOR\*/

line(260,400,260,320);

line(260,320,340,320);

line(340,320,340,400);

/\*Window\*/

line(240,250,270,250);

line(240,250,255,220);

line(255,220,270,250);

line(320,250,350,250);

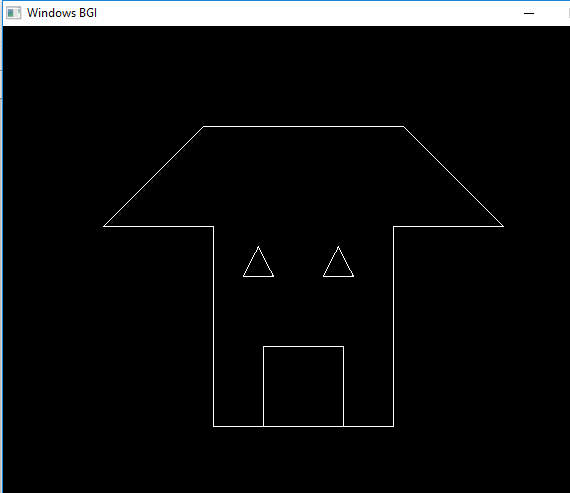
line(320,250,335,220);

line(335,220,350,250);

getch();

closegraph();

}

**OUTPUT:**

# LAB 4.1 WAP TO DRAW FLOOR FILLING

**THEORY:**

The flood fill algorithm has many characters similar to boundary fill. But this method is more suitable for filling multiple colors boundary. When boundary is of many colors and interior is to be filled with one color, we use this algorithm.

In fill algorithm, we start from a specified interior point (x, y) and reassign all pixel values are currently set to a given interior color with the desired color. Using either a 4-connected or 8-connected approaches, we then step through pixel positions until all interior points have been repainted.

**ALGORITHM:**

1. Create a function called as floodFill (x,y,oldcolor,newcolor)

void floodFill(int x,int y,int oldcolor,int newcolor){

if(getpixel(x,y) == oldcolor){

putpixel(x,y,newcolor);

floodFill(x+1,y,oldcolor,newcolor);

floodFill(x,y+1,oldcolor,newcolor);

floodFill(x-1,y,oldcolor,newcolor);

floodFill(x,y-1,oldcolor,newcolor);

}

}

2. Repeat until the polygon is completely filled.

3. Stop.

**SOURCE CODE:**

#include<graphics.h>

void floodFill(int x,int y,int oldcolor,int newcolor){

if(getpixel(x,y) == oldcolor){

putpixel(x,y,newcolor);

floodFill(x+1,y,oldcolor,newcolor);

floodFill(x,y+1,oldcolor,newcolor);

floodFill(x-1,y,oldcolor,newcolor);

floodFill(x,y-1,oldcolor,newcolor);

}

}

//getpixel(x,y) gives the color of specified pixel

int main(){

int gm,gd=DETECT,radius;

int x,y;

printf("Enter x and y positions for circle\n");

scanf("%d%d",&x,&y);

printf("Enter radius of circle\n");

scanf("%d",&radius);

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

circle(x,y,radius);

floodFill(x,y,0,15);

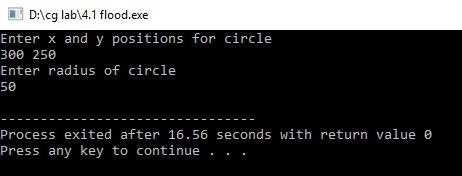
delay(5000);

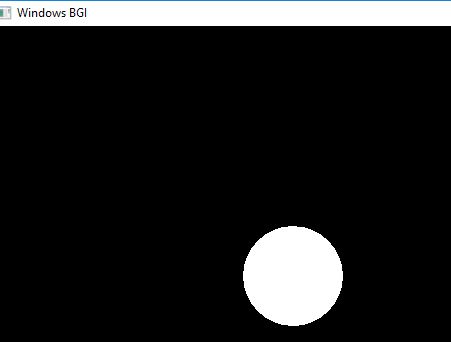
closegraph();

return 0;

}

**OUTPUT:**

****

****

# LAB 4.2 WAP TO DRAW BOUNDARY FILLING

**THEORY:**

Boundary Fill is another seed fill algorithm in which edges of the polygon are drawn. Then starting with some seed any point inside the polygon we examine the neighboring pixels to check whether the boundary pixel is reached. If boundary pixels are not reached, pixels are highlighted and process is continued until boundary pixels are reached.

**ALGORITHM:**

1. Create a function named as boundaryfill with 4 parameters (x,y,f\_color,b\_color).

void boundaryfill(int x,int y,int f\_color,int b\_color)

{

if(getpixel(x,y)!=b\_color && getpixel(x,y)!=f\_color)

{

putpixel(x,y,f\_color);

boundaryfill(x+1,y,f\_color,b\_color);

boundaryfill(x,y+1,f\_color,b\_color);

boundaryfill(x-1,y,f\_color,b\_color);

boundaryfill(x,y-1,f\_color,b\_color);

}

}

//getpixel(x,y) gives the color of specified pixel

2. Call it recursively until the boundary pixels are reached.

3. Stop.

**SOURCE CODE:**

#include<graphics.h>

void boundaryfill(int x,int y,int f\_color,int b\_color){

if(getpixel(x,y)!=b\_color && getpixel(x,y)!=f\_color){

putpixel(x,y,f\_color);

boundaryfill(x+1,y,f\_color,b\_color);

boundaryfill(x,y+1,f\_color,b\_color);

boundaryfill(x-1,y,f\_color,b\_color);

boundaryfill(x,y-1,f\_color,b\_color);

}

}

//getpixel(x,y) gives the color of specified pixe

int main(){

int gm,gd=DETECT,radius;

int x,y;

printf("Enter x and y positions for circle\n");

scanf("%d%d",&x,&y);

printf("Enter radius of circle\n");

scanf("%d",&radius);

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

circle(x,y,radius);

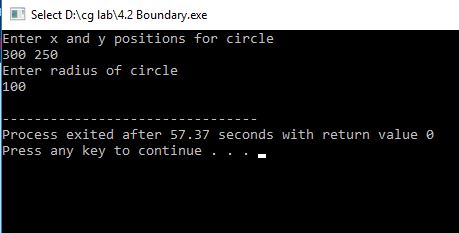
boundaryfill(x,y,4,15);

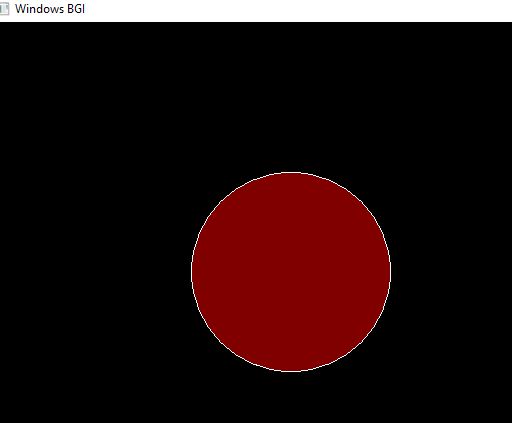
delay(5000);

closegraph();

return 0;

}

**OUTPUT:**



# LAB 5.1 WAP TO DRAW LINE USING BRESENHAM ALGORITHM

**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.

**ALGORITHM:**

Step1: Start Algorithm

Step2: Declare variable x1,x2,y1,y2,d,i1,i2,dx,dy

Step3: Enter value of x1,y1,x2,y2 Where x1,y1are coordinates of starting point And x2,y2 are coordinates of Ending point

Step4: Calculate dx = x2-x1

Calculate dy = y2-y1

Calculate i1=2\*dy

Calculate i2=2\*(dy-dx)

Calculate d=i1-dx

Step5: Consider (x, y) as starting point and x end as maximum possible value of x.

If dx < 0

Then x = x2

y = y2

x end=x1

If dx > 0

Then x = x1

y = y1

x end=x2

Step6: Generate point at (x,y)coordinates

Step7: Check if whole line is generated. If x > = xend Stop.

Step8: Calculate co-ordinates of the next pixel

If d < 0

Then d = d + i1

If d ≥ 0

Then d = d + i2 Increment y = y + 1

Step9: Increment x = x + 1

Step10: Draw a point of latest (x, y) coordinates

Step11: Go to step 7

Step12: End of Algorithm

**SOURCE CODE:**

#include<stdio.h>

#include<graphics.h>

void drawline(int x0, int y0, int x1, int y1){

int dx, dy, p, x, y;

dx=x1-x0;

dy=y1-y0;

x=x0;

y=y0;

p=2\*dy-dx;

while(x<x1){

if(p>=0){

putpixel(x,y,7);

y=y+1;

p=p+2\*dy-2\*dx;

}else{

putpixel(x,y,7);

p=p+2\*dy;}

x=x+1;

}

}

int main(){

int gdriver=DETECT, gmode, error, x0, y0, x1, y1;

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

printf("Enter co-ordinates of first point: ");

scanf("%d%d", &x0, &y0);

printf("Enter co-ordinates of second point: ");

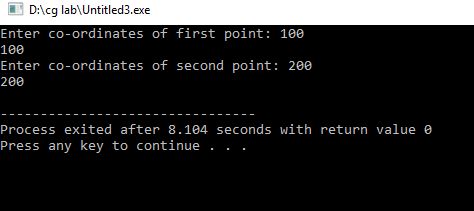
scanf("%d%d", &x1, &y1);

drawline(x0, y0, x1, y1);

return 0;

}

**OUTPUT:**





# LAB 5.2 WAP TO DRAW LINE USING DDA.

**THEORY:**

Digital Differential Analyzer Algorithm or DDA algorithm is the simplest line drawing algorithm. A linear DDA starts by calculating the smaller of dy or dx for a unit increment of the other. A line is then sampled at unit intervals in one coordinate and corresponding integer values nearest the line path are determined for the other coordinate. Considering a line with positive slope, if the slope is less than or equal to 1, we sample at unit x intervals (dx=1) and compute successive y values as

yk+1 = yk + m

xk+1 = xk + 1

Subscript k takes integer values starting from 0, for the 1st point and increases by 1 until endpoint is reached. y value is rounded off to nearest integer to correspond to a screen pixel. For lines with slope greater than 1, we reverse the role of x and y i.e. we sample at dy=1 and calculate consecutive x values as

xk+1 = xk + 1/m

yk+1 = yk + 1

Similar calculations are carried out to determine pixel positions along a line with negative slope. Thus ,if the absolute value of the slope is less than 1, we set dx=1 if x start < x end i.e. the starting extreme point

**ALGORITHM:**

1. Read the end points of a line (x1, y1) & (x2, y2).

2. If the points are same, plot the points & exit (Enter continue with the calculation).

3. dx = | x2-x1 | & dy = | y2-y1 |

4. if abs(dx) ≥ abs(y) then

Steps =dx

Else

Steps =dy

5. x increment =dx / steps.

6. y increment =dy /steps

7. initialize (x, y) with (x1,y1)

x = x1

y = y1

8. plot the rounded coordinate (x, y)

9. initialize counter K=1

10. Start the loop

x = x + x increment

y = y + y increment

Plot the rounded coordinate(x, y)

11. Continue the loop till the counter = steps

12. Stop.

**SOURCE CODE:**

#include <graphics.h>

#include <math.h>

int main( ){

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

int i,gd=DETECT,gm;

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

printf("Enter the value of x1 and y1 : ");

scanf("%f%f",&x1,&y1);

printf("Enter the value of x2 and y2: ");

scanf("%f%f",&x2,&y2);

dx=abs(x2-x1);

dy=abs(y2-y1);

if(dx>=dy)

step=dx;

else

step=dy;

dx=dx/step;

dy=dy/step;

x=x1;

y=y1;

i=1;

while(i<=step){

putpixel(x,y,5);

x=x+dx;

y=y+dy;

i=i+1;

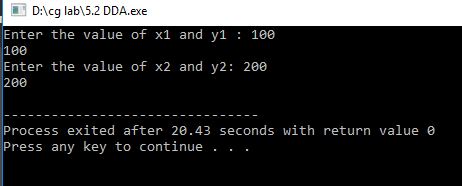
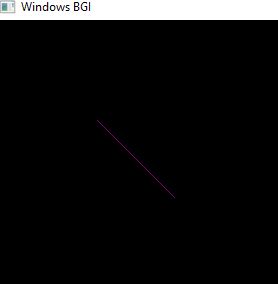
delay(100);

}

closegraph();

}

**OUTPUT:**



# LAB 6 WAP TO DRAW USING MID POINT CIRCLE ALGORITHM.

**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 center.

**ALGORITHM:**

Step 1: Start.

Step 2: First, we allot the center coordinates (p0, q0) as follows

P0 = 0

q0 =r

Step 3: Now, we calculate the initial decision parameter d0 – d0 = 1 – r

Step 4: Assume, the starting coordinates = (pk, qk)

The next coordinates will be (pk+1, qk+1)

Now, we find the next point of the first octant according to the value of the decision parameter (dk).

Step 5: Now, we follow two cases-

Case 1: If dk < 0

then pk+1 =pk + 1

qk+1 =qk

dk+1 = dk + 2 pk+1 + 1

Case 2: If dk >= 0

then pk+1 =pk + 1

qk+1 =qk –1

dk+1 = dk – 2 (qk+1 + 2 pk+1)+ 1

Step 6: If the center coordinate point (p0, q0) is not at the origin (0, 0) then we will drawthe points as follow-

For x coordinate = xc + p0

For y coordinate = yc + q0

Step 7: We repeat step 5 and 6 until we get x>=y.

Step 8: Stop.

**SOURCE CODE:**

#include<stdio.h>

#include<graphics.h>

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

int x = radius;

int y = 0;

int err = 0;

while (x >= y){

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

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

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

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

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

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

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

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

if (err <= 0){

y += 1;

err += 2\*y + 1;

}

if (err > 0){

x -= 1;

err -= 2\*x + 1;

}

}

}

int main(){

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

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

printf("Enter radius of circle: ");

scanf("%d", &r);

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

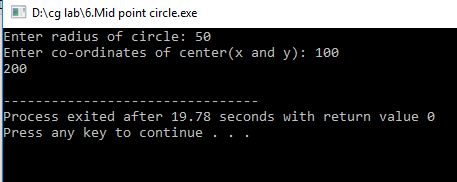
scanf("%d%d", &x, &y);

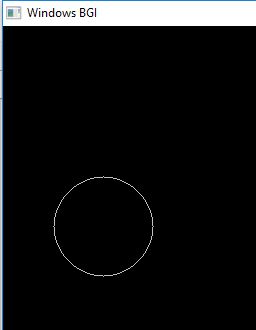
drawcircle(x, y, r);

delay(100);

closegraph();

}

**OUTPUT:**



# LAB 7 WAP TO DRAW USING MID POINT ELLIPSE ALGORITHM.

**THEORY:**

Midpoint ellipse algorithm plots(finds) points of an ellipse on the first quadrant by dividing the quadrant into two regions.

Each point (x, y) is then projected into other three quadrants (-x, y), (x, -y), (-x, -y) i.e. it uses 4-way symmetry.

**ALGORITHM:**

1. Take input radius along x axis and y axis and obtain center of ellipse.

2. Initially, we assume ellipse to be centered at origin and the first point as : (x, y0)=(0, ry).

3. Obtain the initial decision parameter for region 1 as: p10=ry2+1/4rx2-rx 2ry

4. For every xk position in region 1 :

If p1k<0 then the next point along the is (xk+1 , yk) and p1k+1=p1k+2ry2xk+1+ry2

Else, the next point is (xk+1, yk-1 )

And p1k+1=p1k+2ry2xk+1 – 2rx2yk+1+ry2

5. Obtain the initial value in region 2 using the last point (x0, y0) of region 1 as:

p20=ry2(x0+1/2)2+rx2 (y0-1)2-rx2ry2

6. At each yk in region 2 starting at k =0 perform the following task. If p2k>0 the next point is (xk, yk-1) and p2k+1=p2k-2rx2yk+1+rx2

7. Else, the next point is (xk+1, yk -1) and p2k+1=p2k+2ry2xk+1 -2rx2yk+1+rx2

8. Now obtain the symmetric points in the three quadrants and plot the coordinate valueas:

x=x+xc, y=y+yc

9. Repeat the steps for region 1 until 2ry2x>=2rx2y.

**SOURCE CODE:**

#include<stdio.h>

#include<graphics.h>

void ellipse(int xc,int yc,int rx,int ry){

int gm=DETECT,gd;

int x, y, p;

initgraph(&gm,&gd,"C:\\TC\\BGI");

x=0;

y=ry;

p=(ry\*ry)-(rx\*rx\*ry)+((rx\*rx)/4);

while((2\*x\*ry\*ry)<(2\*y\*rx\*rx)){

putpixel(xc+x,yc-y,WHITE);

putpixel(xc-x,yc+y,WHITE);

putpixel(xc+x,yc+y,WHITE);

putpixel(xc-x,yc-y,WHITE);

if(p<0){

x=x+1;

p=p+(2\*ry\*ry\*x)+(ry\*ry);

}else {

x=x+1;

y=y-1;

p=p+(2\*ry\*ry\*x+ry\*ry)-(2\*rx\*rx\*y);

}

}

p=((float)x+0.5)\*((float)x+0.5)\*ry\*ry+(y-1)\*(y-1)\*rx\*rx-rx\*rx\*ry\*ry;

while(y>=0){

putpixel(xc+x,yc-y,WHITE);

putpixel(xc-x,yc+y,WHITE);

putpixel(xc+x,yc+y,WHITE);

putpixel(xc-x,yc-y,WHITE);

if(p>0){

y=y-1;

p=p-(2\*rx\*rx\*y)+(rx\*rx);

} else {

y=y-1;

x=x+1;

p=p+(2\*ry\*ry\*x)-(2\*rx\*rx\*y)-(rx\*rx);

}

}

getch();

closegraph();

}

int main(){

int xc,yc,rx,ry;

printf("Enter Xc=");

scanf("%d",&xc);

printf("Enter Yc=");

scanf("%d",&yc);

printf("Enter Rx=");

scanf("%d",&rx);

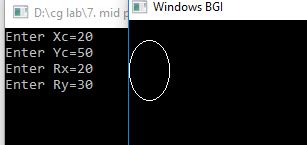
printf("Enter Ry=");

scanf("%d",&ry);

ellipse(xc,yc,rx,ry);

getch();

}

**OUTPUT:**

# LAB 8 WAP TO IMPLEMENT 2D TRANSFORMATION

**THEORY:**

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.

2D Translation is a process of moving an object from one position to another in a two dimensional plane.

**ALGORITHM:**

1.      Start

2.      Initialize the graphics mode.

3.      Construct a 2D object  (use Drawpoly()) e.g. (x,y)

4.      A) Translation

a.       Get the translation value tx, ty

b.      Move the 2d object with tx, ty (x’=x+tx,y’=y+ty)

c.       Plot (x’,y’)

5.      B)  Scaling

a.       Get the scaling value Sx,Sy

b.       Resize the object with Sx,Sy  (x’=x\*Sx,y’=y\*Sy)

c.       Plot (x’,y’)

6.      C) Rotation

a.       Get the Rotation angle

b.      Rotate the object by the angle ф

                    x’=x cos ф -  y sin ф

                    y’=x sin ф  - y cosф

c.       Plot (x’,y’)

**SOURCE CODE:**

#include <graphics.h>

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include<math.h>

int main()

{

int gm;

int gd=DETECT;

int x1,x2,x3,y1,y2,y3,nx1,nx2,nx3,ny1,ny2,ny3,c;

int sx,sy,xt,yt,r;

float t;

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

printf("\t Program for basic transactions");

printf("\n\t Enter the points of triangle");

setcolor(1);

scanf("%d%d%d%d%d%d",&x1,&y1,&x2,&y2,&x3,&y3);

line(x1,y1,x2,y2);

line(x2,y2,x3,y3);

line(x3,y3,x1,y1);

getch();

printf("\n 1.Transaction\n 2.Rotation\n 3.Scalling\n 4.exit");

printf("Enter your choice:");

scanf("%d",&c);

switch(c){

case 1:

printf("\n Enter the translation factor");

scanf("%d%d",&xt,&yt);

nx1=x1+xt;

ny1=y1+yt;

nx2=x2+xt;

ny2=y2+yt;

nx3=x3+xt;

ny3=y3+yt;

line(nx1,ny1,nx2,ny2);

line(nx2,ny2,nx3,ny3);

line(nx3,ny3,nx1,ny1);

getch();

case 2:

printf("\n Enter the angle of rotation");

scanf("%d",&r);

t=3.14\*r/180;

nx1=abs(x1\*cos(t)-y1\*sin(t));

ny1=abs(x1\*sin(t)+y1\*cos(t));

nx2=abs(x2\*cos(t)-y2\*sin(t));

ny2=abs(x2\*sin(t)+y2\*cos(t));

nx3=abs(x3\*cos(t)-y3\*sin(t));

ny3=abs(x3\*sin(t)+y3\*cos(t));

line(nx1,ny1,nx2,ny2);

line(nx2,ny2,nx3,ny3);

line(nx3,ny3,nx1,ny1);

getch();

case 3:

printf("\n Enter the scalling factor");

scanf("%d%d",&sx,&sy);

nx1=x1\*sx;

ny1=y2\*sy;

nx2=x2\*sx;

ny2=y2\*sy;

nx3=x3\*sx;

ny3=y3\*sy;

line(nx1,ny1,nx2,ny2);

line(nx2,ny2,nx3,ny3);

line(nx3,ny3,nx1,ny1);

getch();

case 4:

break;

default:

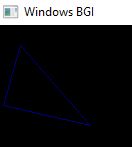
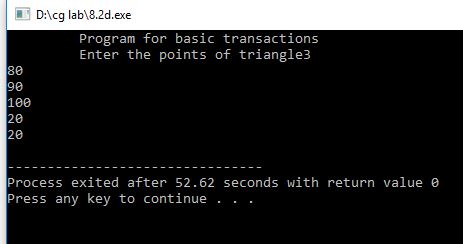
printf("Enter the correct choice");

}

closegraph();

}

**OUTPUT:**

****

# LAB 9 WAP TO IMPLEMENT 3D TRANSFORMATION

**THEORY:**

3-D Transformation is the process of manipulating the view of a three-D object with respect to its original position by modifying its physical attributes through various methods of transformation like Translation, Scaling, Rotation, Shear, etc.

3-D Transformations take place in a three-dimensional plane. 3D Transformations are important and a bit more complex than 2D Transformations. Transformations are helpful in changing the position, size, orientation, shape etc. of the object.

**ALGORITHM:**

1.      Start

2.      Initialize the graphics mode.

3.      Draw a 3D object.

4.      A) Translation

a.       Get the translation value tx, ty

b.      Move the  object with tx, ty (x’=x+tx,y’=y+ty, z’=z+tz)

c.       Plot (x’, y’)

5.      B)  Scaling

a.       Get the scaling value Sx,Sy

b.       Resize the object with Sx,Sy  (x’=x\*Sx,y’=y\*Sy)

c.       Plot (x’,y’)

6.      C) Rotation

a.       Get the Rotation angle

b.      Rotate the object by the angle ф

1.      x’=x cos ф -  y sin ф

2.      y’=x sin ф  - y cosф

c.       Plot (x’,y’)

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void trans();

void scale();

void rotate();

int maxx,maxy,midx,midy;

int main(){

int ch;

int gd=DETECT,gm;

detectgraph(&gd,&gm);

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

printf("\n 1.Translation \n2.Scaling\n 3.Rotation \n 4.exit");

printf("enter your choice");

scanf("%d",&ch);

do{

switch(ch){

case 1 : trans();

getch();

break;

case 2 : scale();

getch();

break;

case 3 : rotate();

getch();

break;

case 4 : break;

}

printf("enter your choice");

scanf("%d",&ch);

} while(ch<4);

}

void trans(){

int x,y,z,o,x1,x2,y1,y2;

maxx=getmaxx();

maxy=getmaxy();

midx=maxx/2;

midy=maxy/2;

bar3d(midx+50,midy-100,midx+60,midy-90,10,1);

printf("Enter translation factor");

scanf("%d%d",&x,&y);

printf("After translation:");

bar3d(midx+x+50,midy-(y+100),midx+x+60,midy-(y+90),10,1);

}

void scale(){

int x,y,z,o,x1,x2,y1,y2;

maxx=getmaxx();

maxy=getmaxy();

midx=maxx/2;

midy=maxy/2;

bar3d(midx+50,midy-100,midx+60,midy-90,5,1);

printf("before translation\n");

printf("Enter scaling factors\n");

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

printf("After scaling\n");

bar3d(midx+(x\*50),midy-(y\*100),midx+(x\*60),midy-(y\*90),5\*z,1);

}

void rotate(){

int x,y,z,o,x1,x2,y1,y2;

maxx=getmaxx();

maxy=getmaxy();

midx=maxx/2;

midy=maxy/2;

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\*sin(o\*3.14/180)+100\*cos(o\*3.14/180);

x2=60\*cos(o\*3.14/180)-90\*sin(o\*3.14/180);

y2=60\*sin(o\*3.14/180)+90\*cos(o\*3.14/180);

printf("After rotation about x axis");

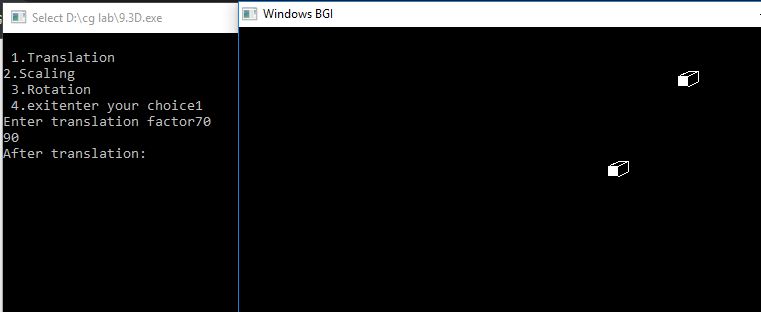
bar3d(midx+50,midy-x1,midx+60,midy-x2,5,1);

printf("After rotation about yaxis");

bar3d(midx+x1,midy-100,midx+x2,midy-90,5,1);

}

**OUTPUT:**



# LAB 10 WAP TO IMPLEMENT SUNDERLAND LINE CLIPPING

**THEORY:**

Cohen Sutherland Algorithm is a line clipping algorithm that cuts lines to portions which are within a rectangular area. It eliminates the lines from a given set of lines and rectangle area of interest (view port) which belongs outside the area of interest and clip those lines which are partially inside the area of interest.

**ALGORITHM:**

**1.** Read 2 end points of line as p1(x1,y1) and p2(x2,y2)

**2.** Read 2 corner points of the clipping window (left-top and right-bottom) as (wx1,wy1) and (wx2,wy2)

**3.** Assign the region codes for 2 endpoints p1 and p2 using following steps:-

initialize code with 0000

Set bit 1 if x<wx1

Set bit 2 if x>wx2

Set bit 3 if y<wy2

Set bit 4 if y>wy1

**4.** Check for visibility of line

1. If region codes for both endpoints are zero then line is completely visible. Draw the line go to step 9.
2. If region codes for endpoints are not zero and logical ANDing of them is also nonzero then line is invisible. Discard the line and move to step 9.
3. If it does not satisfy 4.a and 4.b then line is partially visible.

**5.** Determine the intersecting edge of clipping window as follows:-

1. If region codes for both endpoints are nonzero find intersection points p1’ and p2’ with boundary edges.
2. If region codes for any one end point is non zero then find intersection point p1’ or p2’.

**6.** Divide the line segments considering intersection points.

**7.** Reject line segment if any end point of line appears outside of any boundary.

**8.** Draw the clipped line segment.

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<graphics.h>

#include<dos.h>

typedef struct coordinate{

int x,y;

char code[4];

}PT;

void drawwindow();

void drawline(PT p1,PT p2);

PT setcode(PT p);

int visibility(PT p1,PT p2);

PT resetendpt(PT p1,PT p2);

int main()

{

int gd=DETECT,v,gm;

PT p1,p2,p3,p4,ptemp;

printf("\nEnter x1 and y1\n");

scanf("%d %d",&p1.x,&p1.y);

printf("\nEnter x2 and y2\n");

scanf("%d %d",&p2.x,&p2.y);

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

drawwindow();

delay(500);

drawline(p1,p2);

delay(500);

cleardevice();

delay(500);

p1=setcode(p1);

p2=setcode(p2);

v=visibility(p1,p2);

delay(500);

switch(v)

{

case 0: drawwindow();

delay(500);

drawline(p1,p2);

break;

case 1: drawwindow();

delay(500);

break;

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

p4=resetendpt(p2,p1);

drawwindow();

delay(500);

drawline(p3,p4);

break;

}

delay(5000);

closegraph();

}

void drawwindow(){

line(150,100,450,100);

line(450,100,450,350);

line(450,350,150,350);

line(150,350,150,100);

}

void drawline(PT p1,PT p2){

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

}

PT setcode(PT p) //for setting the 4 bit code

{

PT ptemp;

if(p.y<100)

ptemp.code[0]='1'; //Top

else

ptemp.code[0]='0';

if(p.y>350)

ptemp.code[1]='1'; //Bottom

else

ptemp.code[1]='0';

if(p.x>450)

ptemp.code[2]='1'; //Right

else

ptemp.code[2]='0';

if(p.x<150)

ptemp.code[3]='1'; //Left

else

ptemp.code[3]='0';

ptemp.x=p.x;

ptemp.y=p.y;

return(ptemp);

}

int visibility(PT p1,PT p2)

{

int i,flag=0;

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

{

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

flag=1;

}

if(flag==0)

return(0);

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

{

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

flag='0';

}

if(flag==0)

return(1);

return(2);

}

PT resetendpt(PT p1,PT p2)

{

PT temp;

int x,y,i;

float m,k;

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

x=150;

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

x=450;

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

{

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

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

temp.y=k;

temp.x=x;

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

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

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

return (temp);

}

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

y=100;

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

y=350;

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

{

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

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

temp.x=k;

temp.y=y;

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

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

return(temp);

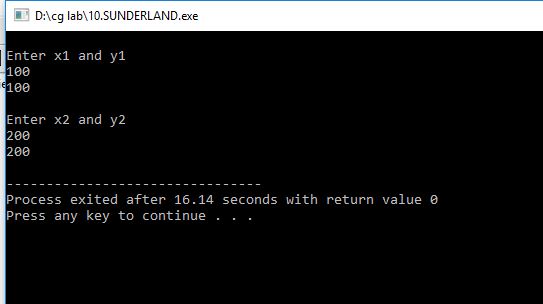
}

else

return(p1);

}

**OUTPUT:**



# LAB 11 WAP TO SHOW BOUNCING BALL

**SOURCE CODE:**

#include<graphics.h>

#include<stdio.h>

int y;

int move(int z){

int x = getmaxx()/2;

while(y+40<=getmaxy()){

circle(x,y,40);

floodfill(x+10,y+10,WHITE);

y++;

delay(10);

cleardevice();

}

while(y>=z){

circle(x,y,40);

floodfill(x+10,y+10,WHITE);

y--;

delay(10);

cleardevice();

}

}

int main(){

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TC\\BGI");

setfillstyle(SOLID\_FILL,RED);

int z=getmaxy()/2;

y=z;

move(z+50);

move(z+100);

move(z+150);

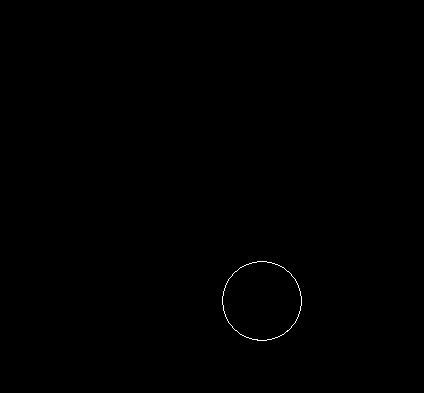
move(z+200);

getch();

closegraph();

}

**OUTPUT:**

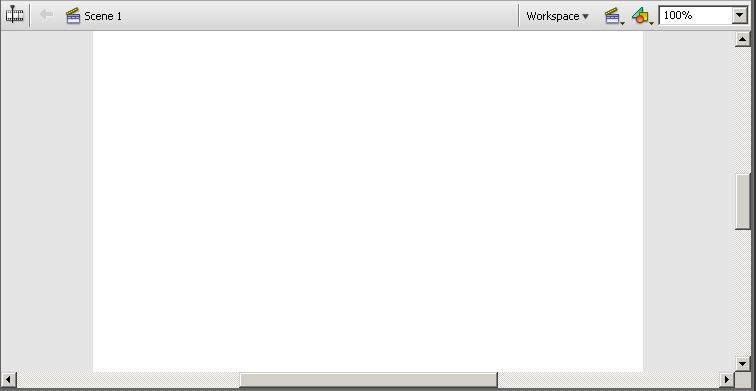


# LAB 12 INTRODUCTION TO FLASH

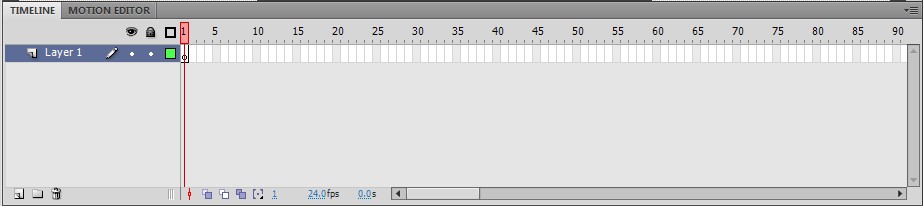
Flash is a powerful graphics-oriented animation and programming environment. Flash has many uses, from simple movie animations to interactive games. This example contains no user interaction so it is not necessary to learn Flash’s programming language – ActionScript, but it will step you through creating a movie that runs for a specific period of time.

.

1. **Start** **All Programs****Adobe** **Adobe Flash Professional CS6**. To make a new movie, click on **File** **New**, and in the dialogue box, select **“ActionScript 3.0”** and click **OK.**
2. On the top of the flash interface that appears on the screen, you will see a classic Windows-style menu bar. Below the menu bar, is the **stage**, which looks like this:



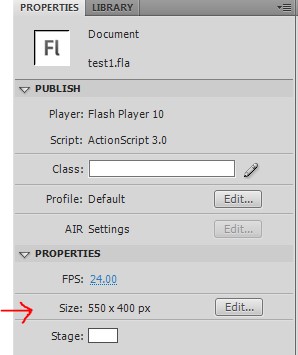
The stage shows the frame of your movie that was selected along the timeline. This is your design workspace. Below this is the **timeline** which looks like this:



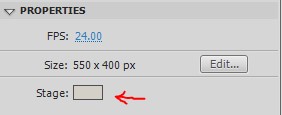
The timeline tells you which file is currently active, which layer is currently active, and at which frame in the movie you are. To the right of the window is a properties window. This is

where you adjust the settings of the different objects you add to your movie. There is also a Library tab with all of the files. To the right of this window is a bar with various tools.

1. The first step in making your movie is to set up your workspace, which is unique to every movie. You should always name your file before you do anything. (**File****Save AS** …) If you know your canvas size, input that now by clicking the button in the **properties** panel.



Next, make the background color similar to the one on your destination. Digital signage monitors in the College of Engineering have a light yellow/brown background, so pick a light yellow by clicking on the **Stage** color palette below the size button.



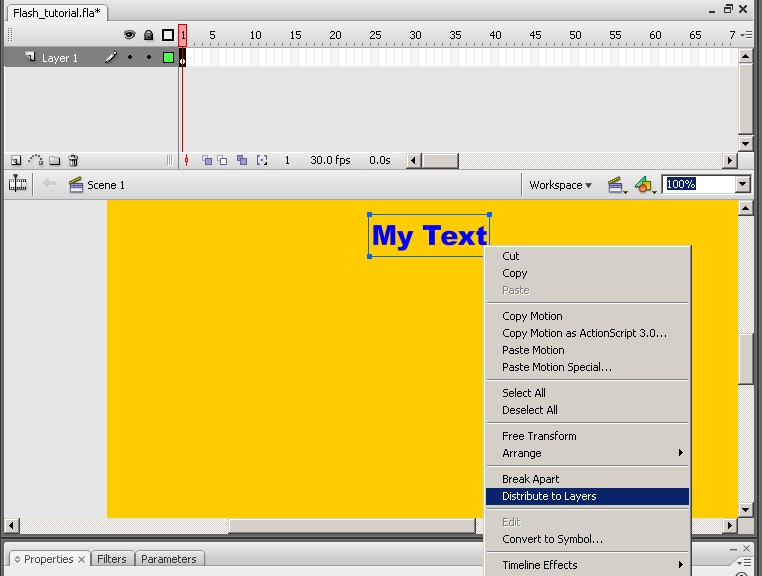
Next, adjust your frame rate by clicking on the blue number to the right of **FPS**. 30 frames per second (fps) makes a nice, smooth movie. There is a trade off to adding more frames though; if you set a movie to 30 fps as opposed to 15, and they are both “x” seconds long, the 30 fps movie will have twice the frames as the 15 fps movie. In theory, this is not a problem, but while actually making the movie, there is a tendency to cut things too short. With this said, 30fps is ideal for viewing, 12 is on the low end and will look choppy, and higher than 30 can

get unmanageable while making the movie. Now that the workspace is set up, we can start designing the first frame.

1. To add text, select the “**T**” icon from the vertical toolbar on the Right side of the window. Click once where you want the text to go, and start typing. You can adjust font, size, color, and other settings in the properties pane while the text (box) is selected.
2. Graphics can simply be added to the stage by dragging them in. If you have many graphics, see **Flash help** to use the image library. You can adjust an image’s proportions using the free **transform tool**, third from the top of the vertical toolbar.

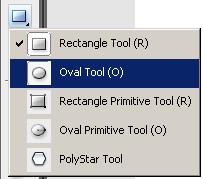
 To resize an image without losing proportions, grab a corner while holding down the shift key, resize, then let up on the mouse before letting up on the shift key

1. Now that you have an idea of how to design in the Flash window, the next step is to animate your objects. In order to keep things simple and not take away from the message trying to be conveyed, make animations simple. They need not do anything more than provide a smooth transition between ideas. There are many ways to animate in Flash. If you want to learn ActionScript, you can code your own animations, giving you the ultimate flexibility in design. For our purposes, motion tweening does a fine job, and is simple to learn and use.
2. To use motion tweening, you need to have the object, or group of objects that will be animated on their own unique layer. Right click the object(s), and select “**distribute to layers**”.

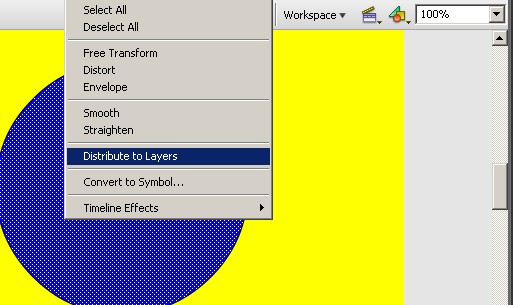


1. Once all the objects are on their own layers, you need to tell Flash how and when you want the object to begin and end. Flash will take care of getting it from beginning to end. The following example shows how to transform a large circle into a small circle.

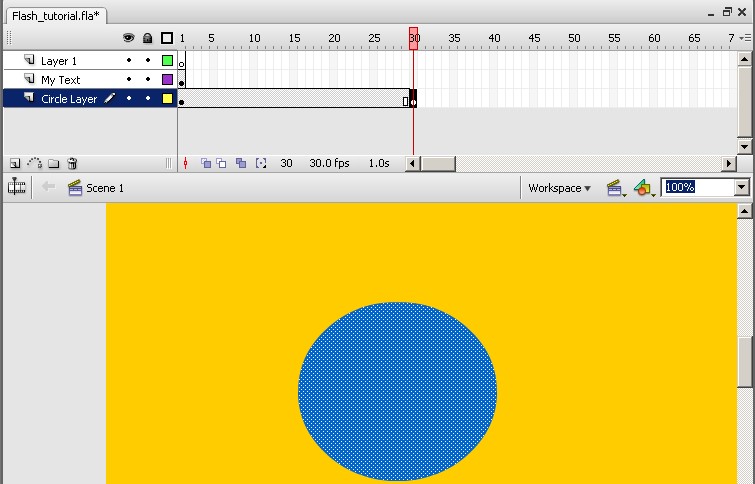
1. Start off by drawing a large circle on its own layer. To do this, select the **oval tool** from the **shapes** menu, ninth from the top on the vertical tool bar on the



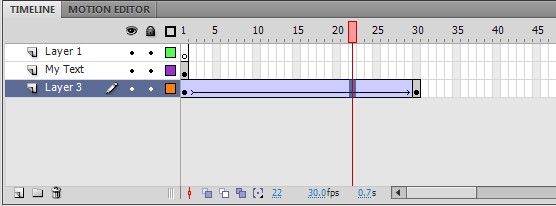
Right. Once the tool is selected, shift-drag to make a large circle. (dimensions of 252x252 would be ideal). Right-click the circle and select **“distribute to layers”** to put the circle in a new layer.



1. You just drew the first frame of the animation. Shapes, lines, and text in Flash need to be converted to symbols before they can be animated. When an object is converted to a symbol, we can create instances of it. This limits the amount of editing that can be done to the object to only simple procedures, like using the free transform tool. To convert an object to a symbol, either right-click the object and select “**convert to symbol**”, or tap the **F8 key**. Click **OK** when the dialogue box shows up. You now have a symbol that can be animated.
2. Depending on how long you want the animation to be, and the amount of fps you chose, click the frame on the timeline where you want the animation to end. Tap the **F5 key**. You will notice that all the frames up to and including the one you clicked turned gray. If you drag the playhead (the red bar over the timeline) over this gray area, you will not see anything change over the duration of those gray frames. Now that you have a static clip you will need to earmark the end as a keyframe. This lets Flash know that you either want to start or end an animation at that point. Click the last frame in the timeline and tap the **F6 key**. You will notice a black dot appear on that block. This means that the particular frame is a keyframe.

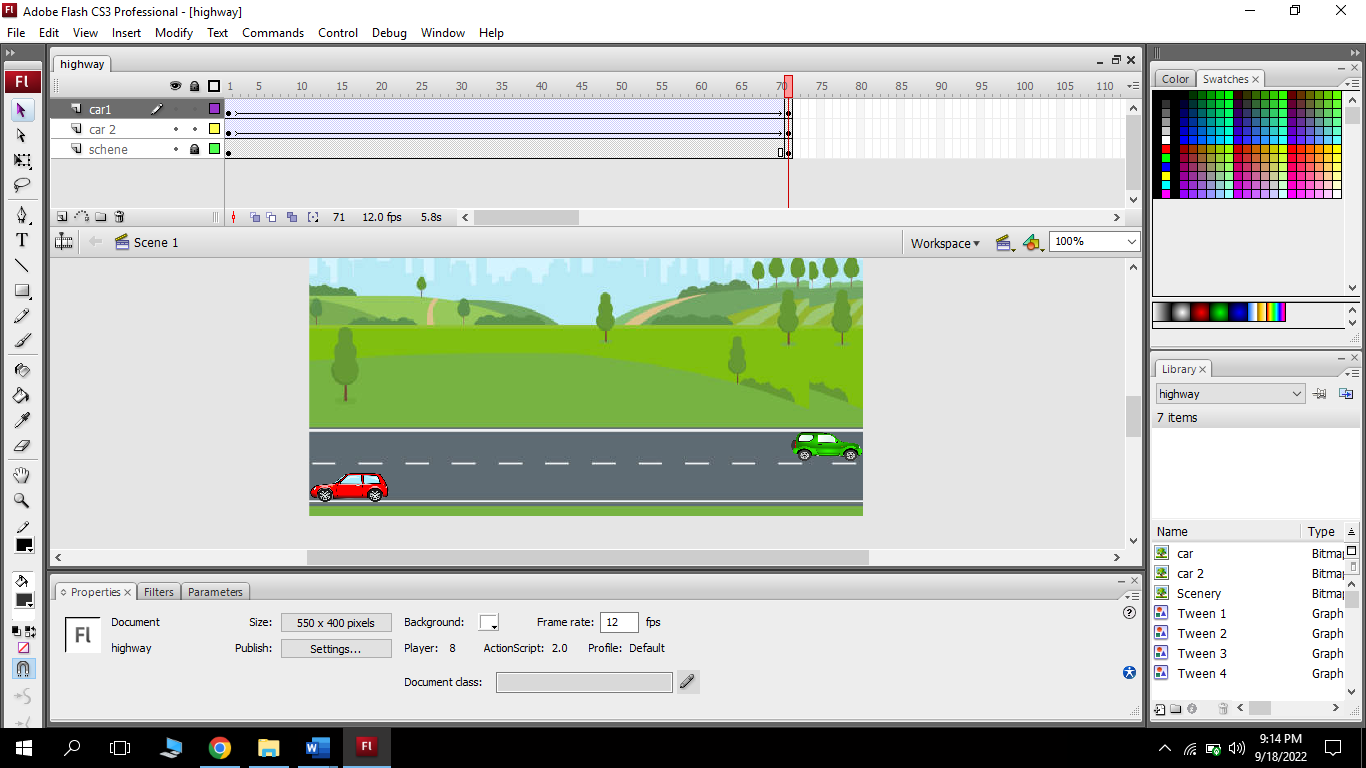


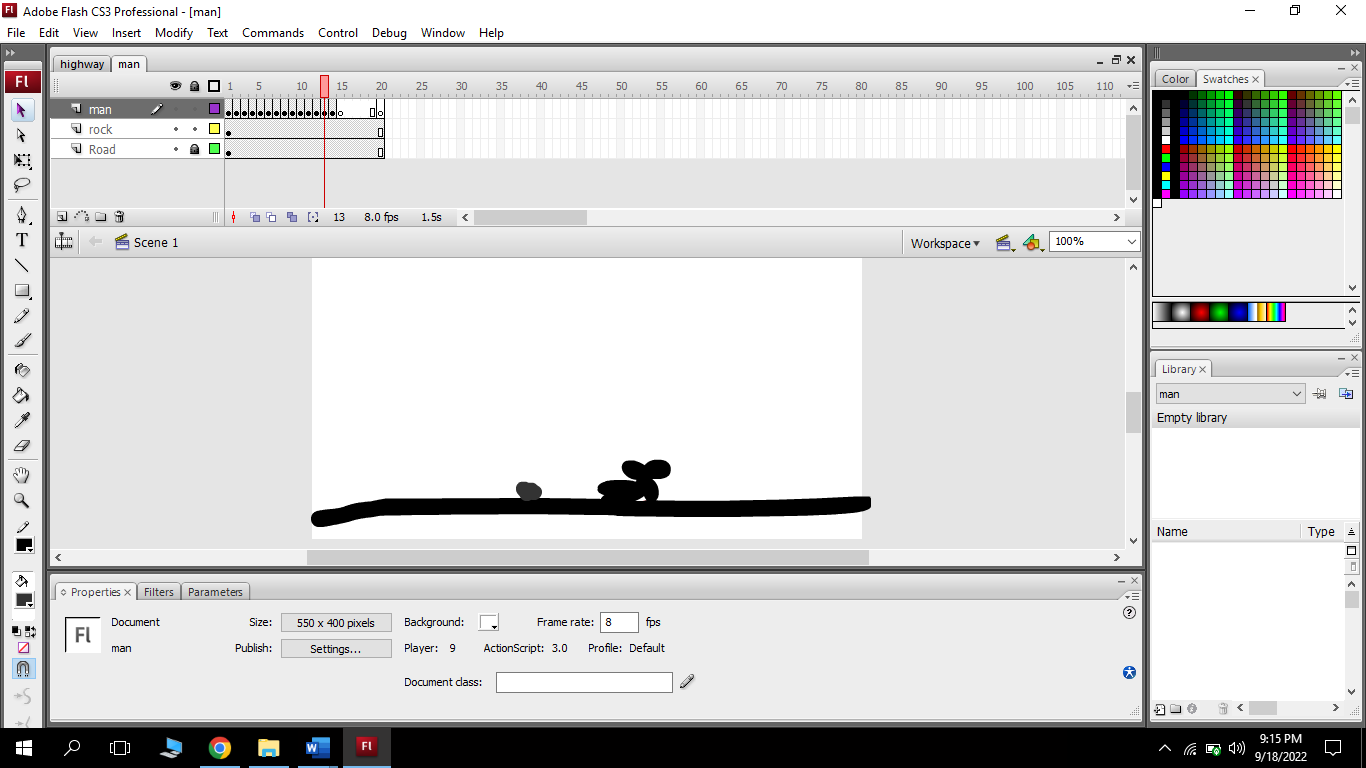
1. Now that the start and endpoint of the animation are defined, you need to make the end different from the beginning. Select the keyframe that you just made, (the one with the black dot) and transform the circle to a smaller one using the **free transform tool**.
2. If you were to play the movie as it is now, the circle would be big, and jump to a small one all of a sudden. To make a smooth animation out of this, which takes up the entire area between the beginning and the keyframe, select any frame before the ending keyframe. In the top menu bar click on **insert**, then **Classic Tween**. The gray box should turn light purple, and you will see an arrow pointing right over the length of the box.



1. To test out the animation, bring the timeline back to the beginning and tap the Enter key. To get a Flash movie that can be played on digital signage in .swf format, select “**test movie**” under the **control menu**, or tap **Ctrl+Enter**. Not only will a finalized movie looping in its own screen, but a movie in .swf format will show up in the same directory of the original movie.
2. Between the free transform tool and motion tweening, you will be able to make many neat-looking animations in Flash. Be creative. If you need inspiration, there are many sites on the internet that show simple Flash animations. Below are some basic guidelines, but the rest is up to you.

**EXAMPLES:**



***Fig: Car Animation***

***Fig: Man hit by stone***