

COMPUTERGRAPHICS AND MULTIMEDIA ETCS 257

Faculty name: Sandeep Tayal

Student name: Syeda Reeha Quasar

Roll No.: 14114802719

Semester: 3C7 (3rd semester)



Maharaja Agrasen Institute of Technology, PSP area, Sector – 22,
Rohini, New Delhi – 110085

(Affiliated to Guru Gobind Singh Indraprastha University, New Delhi)

Introduction to the Lab

1.1 Lab Objective

This course aims to develop the understanding of concepts and algorithms of computer graphics using C Programming. Also, this helps in understanding the working of MAYA tool for 2D/3D objects and their transformation using MAYA tool.

1.2 Course Outcomes

On successful completion of this Course, students should be able to:

257.1. Understand the use of C Graphics Library for writing the programs.

257.2. Implantation of scan conversion algorithms using C Programming.

257.3. Implementing the concept of 2D/3D transformation,

257.4. Implementing the concept of World & View Coordinate System and Clipping Algorithms.

257.5. Understand the object modeling and transformation using MAYA tool.

Index

Exp. no	Experiment Name	Date of performance	Date of checking	Marks	Signature
1.	a) Study and prepare list of Graphic functions. b) Write a C program to make a hut and car using built-in graphic function.	25 – 08 - 2020	30 – 08 - 2020	9	
2.	DDA Algorithm Write a C program to draw a line using DDA algorithm.	06 – 09 - 2020	07 – 09 - 2020	10	
3.	Bresenham's Algorithm Write a C program to draw a line using Bresenham's algorithm.	06 – 09 - 2020	11 – 09 - 2020	9	
4.	Midpoint Circle Generation Algorithm To implement midpoint circle generation algorithm or bresenham's circle algorithm for drawing a circle of given center (x, y) and radius r.	09 – 09 - 2020	11 – 09 - 2020	10	
5.	BRESENHAM'S CIRCLE DRAWING ALGORITHM Implementation of Bresenham's circle drawing algorithm.	15 – 09 - 2020	20 – 09 - 2020	10	
6A.	Write C Programs for the implementation of 2D transformations.	22 – 09 - 2020	03 – 10 - 2020	10	
6B.	Write C Programs for the implementation of 2D and 3D transformations.	29 – 09 - 2020	03 – 10 - 2020	10	
7.	Write a C program to demonstrate Cohen	27 – 10 - 2020	25 – 11 - 2020	10	

	Sutherland line clipping algorithm.				
8.	Write a C program to draw 4 point Bezier Curve.	27 – 10 - 2020	25 – 11 - 2020	10	
9.	Using Flash/Maya perform different operations (rotation, scaling, move etc.) on objects.	03 – 12 - 2020	03 – 12 - 2020	10	
10.	To bounce a ball using animation.	03 – 12 - 2020	03 – 12 - 2020	10	
Beyond Syllabus					
1.	Write a program to draw a car using inbuild graphics function and translate it from bottom left corner to right bottom corner of screen (Animation).	3 – 11 - 2020	24 – 11 - 2020	10	
2.	Write a program to rotate a circle (alternatively inside and outside) around the circumference of another circle (animation).	3 – 11 - 2020	24 – 11 - 2020	10	
3.	To Write a program in C to draw a Rainbow.	03 – 12 - 2020	03 – 12 - 2020	10	
4.	To Write a program in C to display a digital and analog clock displaying current time.	03 – 12 - 2020	03 – 12 - 2020	10	
5.	Created a program using C. A man walking in Rain.	03 – 12 - 2020	03 – 12 - 2020	10	



EXPERIMENT - 1

Computer Graphics and Multimedia

- a) Study and prepare list of Graphic functions.
- b) Write a C program to make a hut and car using built-in graphic function.

syeda reeha quasar

14114802719

3C7

EXPERIMENT - 1

Aim: a) Study and prepare list of Graphic functions.

b) Write a C program to make a hut and car using built-in graphic function.

Theory:

The graphics. h header file provides access to a simple graphics library that makes it possible to draw lines, rectangles, ovals, arcs, polygons, images, and strings on a graphical window.

Functions:

initGraphics() or initwindow(): Creates the graphics window on the screen.

Syntax - initGraphics(width, height), initwindow(width, height)

1. Putpixel

Purpose:-Putpixel function is to draw the pixel on the screen. Pixel is small dot on the screen.

Syntax:-putpixel(x co-ordinate, y co-ordinate,COLOR);

Example: – putpixel(100,100,BLUE);

2. SetbkColor

Purpose:-Setbkcolor function is used to set background color of the screen.

Syntax:-setbkcolor(COLOR);

Example:-setbkcolor(RED);

3. Setlinestyle

Purpose:-setlinestyle function is used to set the current line style, width and pattern

Syntax:-setlinestyle(linestyle, pattern, thickness);

Example:-setlinestyle(SOLID_LINE,1,2);

4. Setcolor

Purpose:-setcolor is to set color of the objects which is to be drawn after this setcolor line.

Syntax:-setcolor(COLOR);

Example:-setcolor(RED);

5. Rectangle:-

Purpose:- Rectangle function is used to draw the rectangle on the screen. X1,y1 are the lower left co-ordinates of the rectangle and the x2,y2 are the upper right co-ordinates of the rectangle.

Syntax:- rectangle(x1,,y1,x2,y2);

Example:- rectangle(100,100,200,200);

6. Textheight

Purpose:-textheight returns the height of a string in pixels.

Syntax:-textheight(String);

Example:-i=textheight("HELLO");

7. Textwidth

Purpose:-textwidth returns the width of a string in pixels

Syntax:-textwidth(String);

Example:-i=textwidth("HELLO");

8. Getx

Purpose:-getx returns the current position's of x o-ordinate

Syntax:-getx();

Example:-x=getx();

9. Gety

Purpose:-gety returns the current position's of y co-ordinate

Syntax:-gety();

Example:-y=gety();

10. Getmaxx

Purpose:-getmaxxreturns the maximum x co-ordinate on the screen

Syntax:-getmaxx();

Example:-maxx=getmaxx();

11. Getmaxy

Purpose:-getmaxy returns the maximum y co-ordinate on the screen

Syntax:-getmaxy();

Example:-maxy=getmaxy();

12. Line

Purpose:-Line function is used to draw the line on the screen.

Syntax: line(x1,y1,x2,y2);

Example:-line(100,100,200,100);

13. Closegraph

Purpose:-closegraph function shut down the graphic system

Syntax:-closegraph();

Example:-closegraph();

14. Moveto

Purpose:-moveto function moves current cursor position on the screen

Syntax:-moveto(x co-ordinate, y co-ordinate);

Example:-moveto(getmaxx/2, getmaxy/2);

15. Settextstyle

Purpose:-settextstyle sets the current text characteristics like font, direction and size

Syntax:-settextstyle(font, direction size);

Example:-settextstyle(1,1,10);

16. Circle

Purpose: Circle function is used to draw the circle on the screen

Syntax:- circle(x,y,radius);

Example:circle(100,100,50);

17. Cleardevice

Purpose: cleardevice function is used to clear the contents or graphic images on the screen in graphics mode.

Syntax:cleardevice();

Example:cleardevice();

18. Outtextxy

Purpose: outtextxy function is used to print the text on the screen in graphics mode.

Syntax:outtext(x,y,text);

Example:-outtextxy(100,100,"HELLO");

19. Sector

Purpose:sector function draws and fills an elliptical pie slice.

Syntax:sector(x, y, starting angle, ending angle, xradius, yradius);

Example:sector(100,100,45 135 100 50);

20. Arc

Purpose:arc draws the arc on the screen, arc is a part of the circle

Syntax:arc(x, y, starting angle, ending angle, radius);

Example:arc(100,100,90,180,50);

21. Setfillstyle

Purpose: setfillstyle is used to set the color and style to be filled in the object using the flood fill method.

Syntax:setfillstyle(STYLE, COLOR);

Example:setfillstyle(1,RED)

22. Floodfill

Purpose:floodfill function is used to fill the color in the object, object may be circle, rectangle or any other closed image.

Syntax:floodfill(x,y,boundary color);

Example:floodfill(100,100,BLUE);

23. Ellipse

Purpose:ellipse function is used to draw the ellipse on the screen.

Syntax:ellipse(x, y, starting angle, ending angle, xradius, yradius);

Example:ellipse(100,100,90,200,20,20);

24. Outtext

Purpose:outtext function is used to display the text on the screen, using this function text is display in the current position.

Syntax:outtext(String);

Example:outtex("HELLO");

25. Getcolor

Purpose:getcolor returns the current drawing color.

Syntax:getcolor();

Example:intclr = getcolor();

26. Getpixel

Purpose:getpixel gets the color of a specified pixel.

Syntax:getpixel(x,y);

Example: color=getpixel(100,100);

Drawing Functions

For drawing arc:

```
void arc( int x, int y, int stangle, int endangle, int radius );
```

for drawing bar:

```
void bar( int left, int top, int right, int bottom );
```

for drawing 3d bar:

```
void bar3d( int left, int top, int right, int bottom, int depth, int topflag );
```

for drawing a circle:

```
void circle( int x, int y, int radius );
```

for clearing:

```
void cleardevice( );
```

```
void clearviewport( );
```

for drawing polygon:

```
void drawpoly(int n_points, int* points);
```

```
void fillpoly(int n_points, int* points);
```

for drawing ellipse:

```
void ellipse( int x, int y, int stangle, int endangle, int xradius, int yradius );  
void fillellipse( int x, int y, int xradius, int yradius );
```

for filling:

```
void floodfill( int x, int y, int border );
```

for making line:

```
void line( int x1, int y1, int x2, int y2 );
```

```
void linerel( int dx, int dy );
```

```
void lineto( int x, int y );
```

```
void pieslice( int x, int y, int stangle, int endangle, int radius );
```

```
void putpixel( int x, int y, int color );
```

for drawing rectangle:

```
void rectangle( int left, int top, int right, int bottom );
```

for drawing sector:

```
void sector( int x, int y, int stangle, int endangle, int xradius, int yradius );
```

Source Code:

CAR

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

```
#include <conio.h>
```

```
#include <graphics.h>
```

```
main()
```

```
{
```

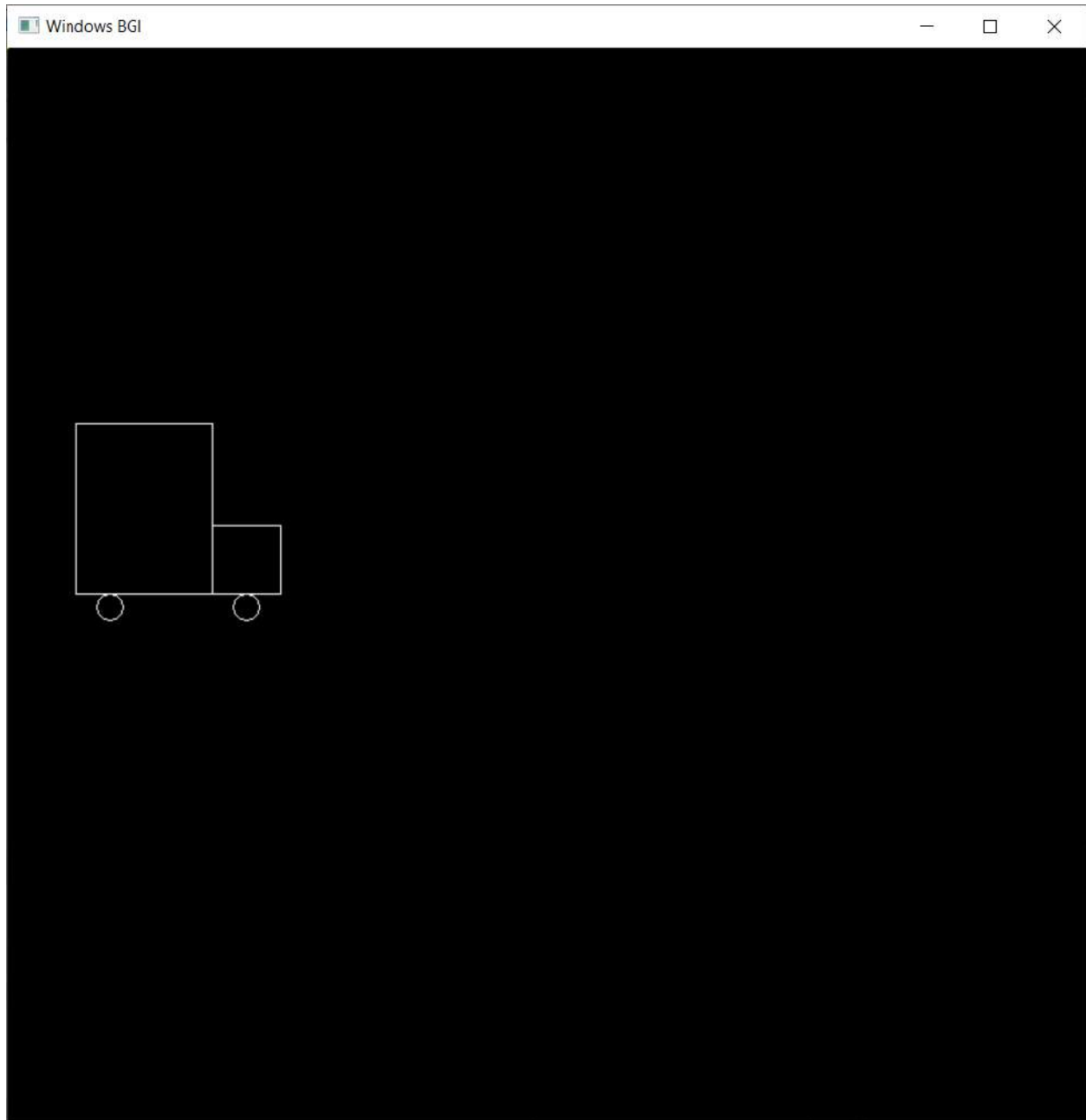
```
    initwindow(800, 800);
```

```
    rectangle(50,275,150,400);
```

```
    rectangle(150,350,200,400);
```

```
circle(75,410,10);  
circle(175,410,10);  
getch();  
}
```

OUTPUT



HUT

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

```
#include <conio.h>
```

```
#include <graphics.h>
```

```
main()
```

```
{
```

```
    initwindow(800, 800);
```

```
    rectangle(150,180,250,300);
```

```
    rectangle(250,180,420,300);
```

```
    rectangle(180,250,220,300);
```

```
    line(200,100,150,180);
```

```
    line(200,100,250,180);
```

```
    line(200,100,370,100);
```

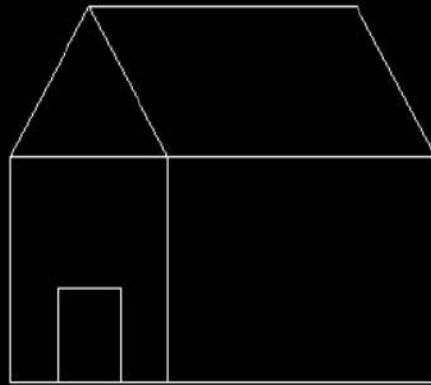
```
    line(370,100,420,180);
```

```
    getch();
```

```
}
```

HUT

Windows BGI



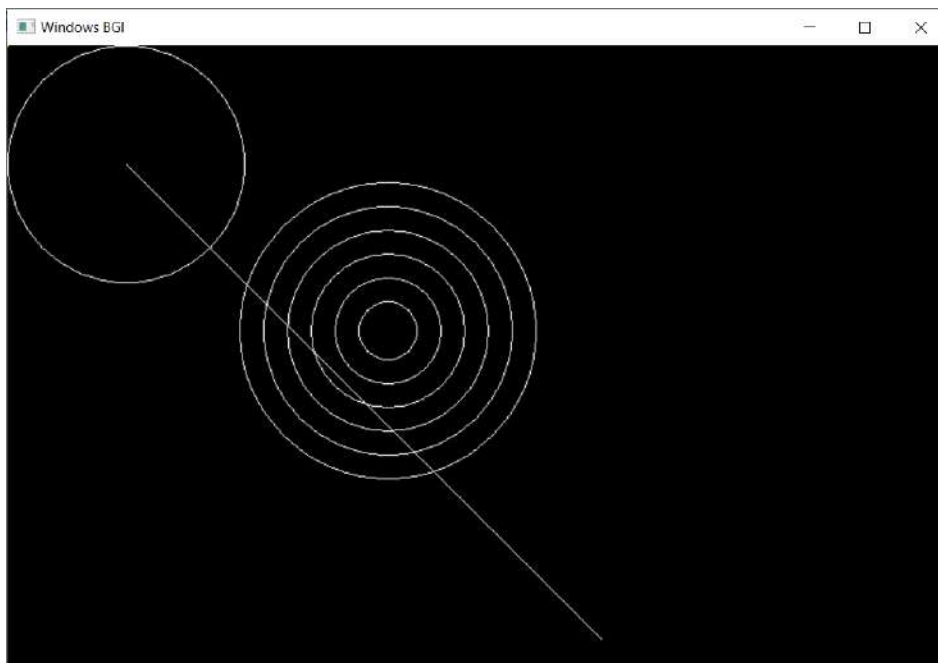
Concentric circle and lines:

```
#include <stdio.h>

#include <conio.h>

#include <graphics.h>

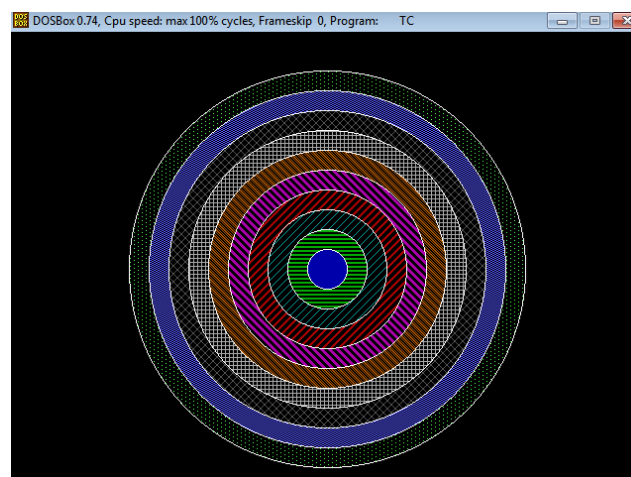
main()
{
    initwindow(800, 800);
    line(100, 100, 500, 500);
    circle(100, 100, 100);
    int x = 320, y = 240, radius;
    for ( radius = 25; radius <= 125 ; radius = radius + 20)
        circle(x, y, radius);
    getch();
}
```



```

#include<graphics.h>
#include<conio.h>
void main()
{
    intgd=DETECT, gm, i, x, y;
    initgraph(&gd, &gm, "C:\\\\TC\\\\BGI");
    x=getmaxx()/3;
    y=getmaxx()/3;
    setbkcolor(WHITE);
    setcolor(BLUE);
    for(i=1;i<=8;i++)
    {
        setfillstyle(i,i);
        delay(20);
        circle(x, y, i*20);
        floodfill(x-2+i*20,y,BLUE);
    }
    getch();
    closegraph();
}

```



VIVA QUESTIONS:

1. What is the use of `initgraph()` and `closegraph()` function?

ANS. `initgraph()` function is used to enter in the graphics mode and `closegraph()` function is used to exit from the graphics mode and also enter in the text mode .

2. Why do we need to use `closegraph()` function after `getch()` ?

ANS. `Getch()` helps us to wait until a key is pressed, `closegraph()` function closes the graphics mode, and finally return statement returns a value 0 to main indicating successful execution of the program. `Getch` is used to hold the output screen and wait until user gives any type of input(i.e. Until user press any key) so that they can read the character and due to this we able to see the output on the screen.

3. Which parameters are used to find the resolution of the screen?

ANS. Pixels are the parameter for measuring the resolution.

4. How is `putpixel()` different from `getpixel()`?

ANS. Function `getpixel` returns the color of pixel present at point(x, y). Function `putpixel` plots a pixel at a point(x, y) of the specified color.

5. Explain various other graphic functions.

ANS.

`Putpixel`

Purpose:-`Putpixel` function is to draw the pixel on the screen. Pixel is small dot on the screen.

Syntax:-`putpixel(x co-orinate, y co-ordinate,COLOR);`

Example: – `putpixel(100,100,BLUE);`

`SetbkColor`

Purpose:-`Setbkcolor` function is used to set background color of the screen.

Syntax:-`setbkcolor(COLOR);`

Example:-`setbkcolor(RED);`

Setlinestyle

Purpose:-setlinestyle function is used to set the current line style, width and pattern

Syntax:-setlinestyle(linestyle, pattern, thickness);

Example:-setlinestyle(SOLID_LINE,1,2);

Setcolor

Purpose:-setcolor is to set color of the objects which is to be drawn after this setcolor line.

Syntax:-setcolor(COLOR);

Example:-setcolor(RED);

Rectangle:-

Purpose:- Rectangle function is used to draw the rectangle on the screen. X1,y1 are the lower left co-ordinates of the rectangle and the x2,y2 are the upper right co-ordinates of the rectangle.

Syntax:- rectangle(x1,,y1,x2,y2);

Example:- rectangle(100,100,200,200);

Textheight

Purpose:-textheight returns the height of a string in pixels.

Syntax:-textheight(STRING);

Example:-i=textheight("HELLO");

Textwidth

Purpose:-textwidth returns the width of a string in pixels

Syntax:-textwidth(STRING);

Example:-i=textwidth("HELLO");

Getx

Purpose:-getx returns the current position's of x o-ordinate

Syntax:-getx();

Example:-x=getx();

Gety

Purpose:-gety returns the current position's of y co-ordinate

Syntax:-gety();

Example:-y=gety();

Getmaxx

Purpose:-getmaxxreturns the maximum x co-ordinate on the screen

Syntax:-getmaxx();

Example:-maxx=getmaxx();

Getmaxy

Purpose:-getmaxy returns the maximum y co-ordinate on the screen

Syntax:-getmaxy();

Example:-maxy=getmaxy();

Line

Purpose:-Line function is used to draw the line on the screen.

Syntax: line(x1,y1,x2,y2);

Example:-line(100,100,200,100);

Closegraph

Purpose:-closegraph function shut down the graphic system

Syntax:-closegraph();

Example:-closegraph();

Moveto

Purpose:-moveto function moves current cursor position on the screen

Syntax:-moveto(x co-ordinate, y co-ordinate);

Example:-moveto(getmaxx/2, getmaxy/2);

Settextstyle

Purpose:-settextstyle sets the current text characteristics like font, direction and size

Syntax:-settextstyle(font, direction size);

Example:-settextstyle(1,1,10);

Circle

Purpose: Circle function is used to draw the circle on the screen

Syntax:- circle(x,y,radius);

Example:circle(100,100,50);

Cleardevice

Purpose: cleardevice function is used to clear the contents or graphic images on the screen in graphics mode.

Syntax:cleardevice();

Example:cleardevice();

Outtextxy

Purpose: outtextxy function is used to print the text on the screen in graphics mode.

Syntax:outtext(x,y,text);

Example:-outtextxy(100,100,"HELLO");

Sector

Purpose:sector function draws and fills an elliptical pie slice.

Syntax:sector(x, y, starting angle, ending angle, xradius, yradius);

Example:sector(100,100,45 135 100 50);

Arc

Purpose:arc draws the arc on the screen, arc is a part of the circle

Syntax:arc(x, y, starting angle, ending angle, radius);

Example:arc(100,100,90,180,50);

Setfillstyle

Purpose:setfillstyle is used to set the color and style to be filled in the object using the flood fill method.

Syntax:setfillstyle(STYLE, COLOR);

Example:setfillstyle(1,RED)

Floodfill

Purpose:floodfill function is used to fill the color in the object, object may be circle, rectangle or any other closed image.

Syntax:floodfill(x,y,boundary color);

Example:floodfill(100,100,BLUE);

Ellipse

Purpose:ellipse function is used to draw the ellipse on the screen.

Syntax:ellipse(x, y, starting angle, ending angle, xradius, yradius);

Example:ellipse(100,100,90,200,20,20);

Outtext

Purpose:outtext function is used to display the text on the screen, using this function text is display in the current position.

Syntax:outtext(STRING);

Example:outtex("HELLO");

Getcolor

Purpose:getcolor returns the current drawing color.

Syntax:getcolor();

Example:intclr = getcolor();

Getpixel

Purpose: getpixel gets the color of a specified pixel.

Syntax: getpixel(x,y);

Example: color=getpixel(100,100);



EXPERIMENT - 2

COMPUTER GRAPICS AND MULTIMEDIA

DDA Algorithm

Write a C program to draw a line using DDA algorithm.

Syeda Reeha Quasar

14114802719

Group - 3C7

EXPERIMENT – 2

AIM:

Write a C program to draw a line using DDA algorithm.

THEORY:

DDA stands for Digital Differential Analyzer. It is an incremental method of scan conversion of line. In this method calculation is performed at each step but by using results of previous steps.

Advantage:

1. It is a faster method than method of using direct use of line equation.
2. This method does not use multiplication theorem.
3. It allows us to detect the change in the value of x and y, so plotting of same point twice is not possible.
4. This method gives overflow indication when a point is repositioned.
5. It is an easy method because each step involves just two additions.

Disadvantage:

1. It involves floating point additions rounding off is done. Accumulations of round off error cause accumulation of error.
2. Rounding off operations and floating-point operations consumes a lot of time.
3. It is more suitable for generating line using the software. But it is less suited for hardware implementation.

DDA Algorithm:

Step1: Start Algorithm

Step2: Declare $x_1, y_1, x_2, y_2, dx, dy, x, y$ as integer variables.

Step3: Enter value of x_1, y_1, x_2, y_2 .

Step4: Calculate $dx = x_2 - x_1$

Step5: Calculate $dy = y_2 - y_1$

Step6: If $ABS(dx) > ABS(dy)$
Then $step = abs(dx)$
Else

Step7: $x_{inc} = dx / step$
 $y_{inc} = dy / step$

```
assign x = x1  
assign y = y1
```

Step8: Set pixel (x, y)

Step9: $x = x + x_{inc}$
 $y = y + y_{inc}$
Set pixels (Round (x), Round (y))

Step10: Repeat step 9 until $x = x_2$

Step11: End Algorithm

Drawing line using DDA Algorithm :

CODE

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

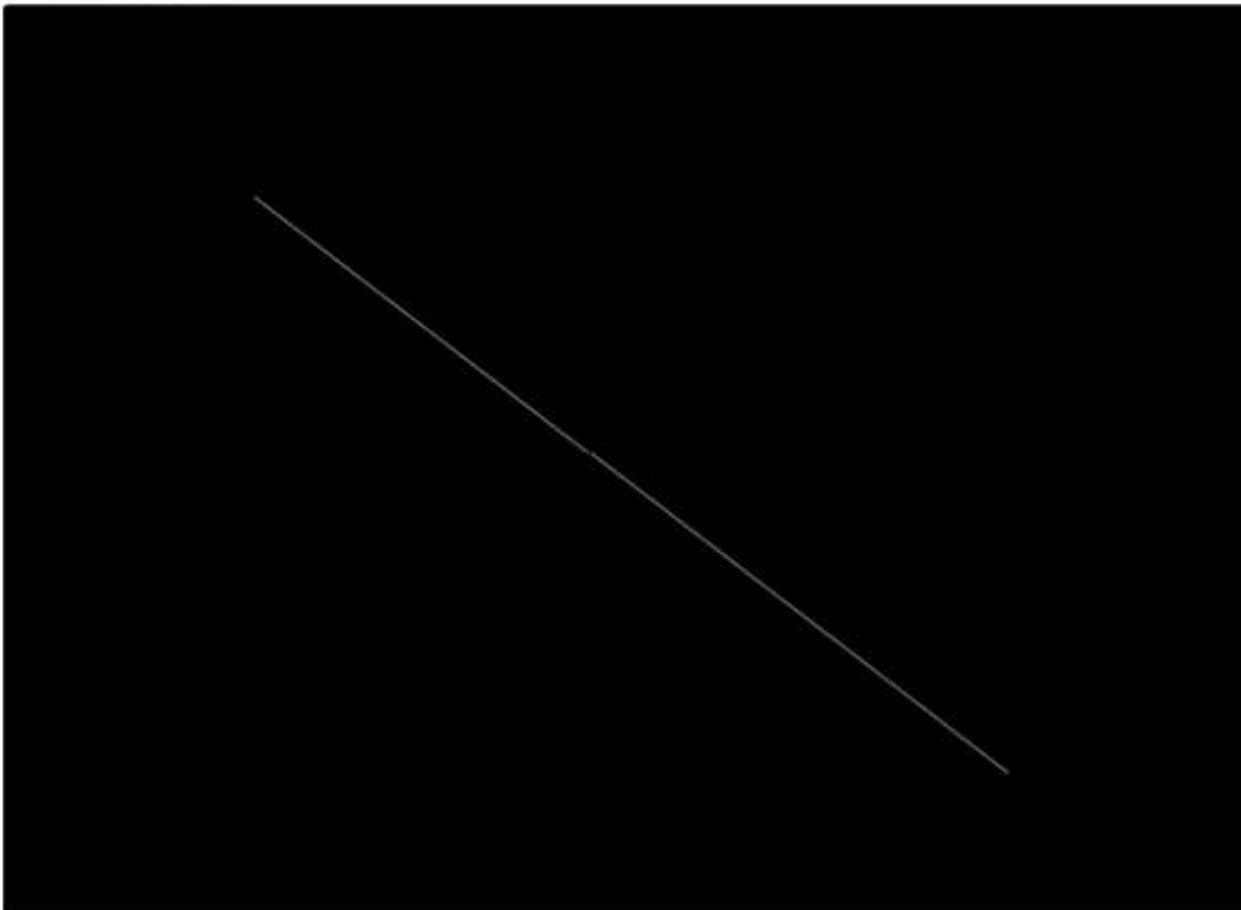
```
Dda_function(int x0,int y0,int x1,int y1,int clr)
```

```
{  
    int i;  
    float x,y,dx,dy ,steps;  
  
    dx=x1-x0;  
    dy=y1-y0;  
    if(abs(dx)>=abs(dy)){  
        steps=dx;  
    }  
    else{  
        steps=dy;  
    }  
    dx=dx/steps;  
    dy=dy/steps;  
    x=x0;  
    y=y0;
```



```
i=1;
while(i<=steps){
    putpixel(x,y,clr);
    x+=dx;
    y+=dy;
    i=i+1;
}
}
main(){
    initwindow(800,800);
    Dda_function(200, 200, 600, 650, 1);
    getch();
    closegraph();
}
```

Windows BGI



Source Code:

```
// C program for drawing a chair
```

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

```
#include<conio.h>
```

```
#include<graphics.h>
```

```
Dda_function(int x0,int y0,int x1,int y1,int clr)
```

```
{
```

```
    int i;
```

```
    float x,y,dx,dy ,steps;
```

```
    dx=x1-x0;
```

```
    dy=y1-y0;
```

```
    if(abs(dx)>=abs(dy)){
```

```
        steps=dx;
```

```
    }
```

```
    else{
```

```
        steps=dy;
```

```
    }
```

```
    dx=dx/steps;
```

```
    dy=dy/steps;
```

```
    x=x0;
```

```
    y=y0;
```

```
    i=1;
```

```
    while(i<=steps){
```

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

```
        x+=dx;
```

```
        y+=dy;
```

```
        i=i+1;
```

```
    }
```

```
}
```

```
main(){
```

```
    initwindow(800,800);
```

```
    // rest
```

```
    Dda_function(200, 200, 400, 150, 1); //top
```

```
    Dda_function(200, 200, 200, 400, 1); //l line
```

```
    Dda_function(400, 150, 400, 350, 1); //r line
```

```
    Dda_function(200, 400, 400, 350, 1); //join
```

```
    // seat
```

```
    Dda_function(200, 400, 350, 450, 1);
```

```
    Dda_function(400, 350, 550, 400, 1);
```

```
    Dda_function(350, 450, 550, 400, 1);
```

```
    // legs
```

```
    Dda_function(200, 400, 200, 500, 1);
```

```
    Dda_function(210, 410, 210, 510, 1);
```

```
    Dda_function(200, 500, 210, 510, 1);
```

```
    Dda_function(400, 440, 400, 520, 1);
```

```
    Dda_function(410, 440, 410, 520, 1);
```

```
    Dda_function(400, 520, 410, 520, 1);
```

```
    Dda_function(340, 450, 340, 590, 1);
```

```
    Dda_function(350, 450, 350, 590, 1);
```

```
    Dda_function(340, 590, 350, 590, 1);
```

```
    Dda_function(550, 400, 550, 500, 1);
```

```
    Dda_function(540, 400, 540, 500, 1);
```

```
    Dda_function(540, 500, 550, 500, 1);
```

```
    //c
```

```
    Dda_function(500, 200, 500, 300, 3);
```

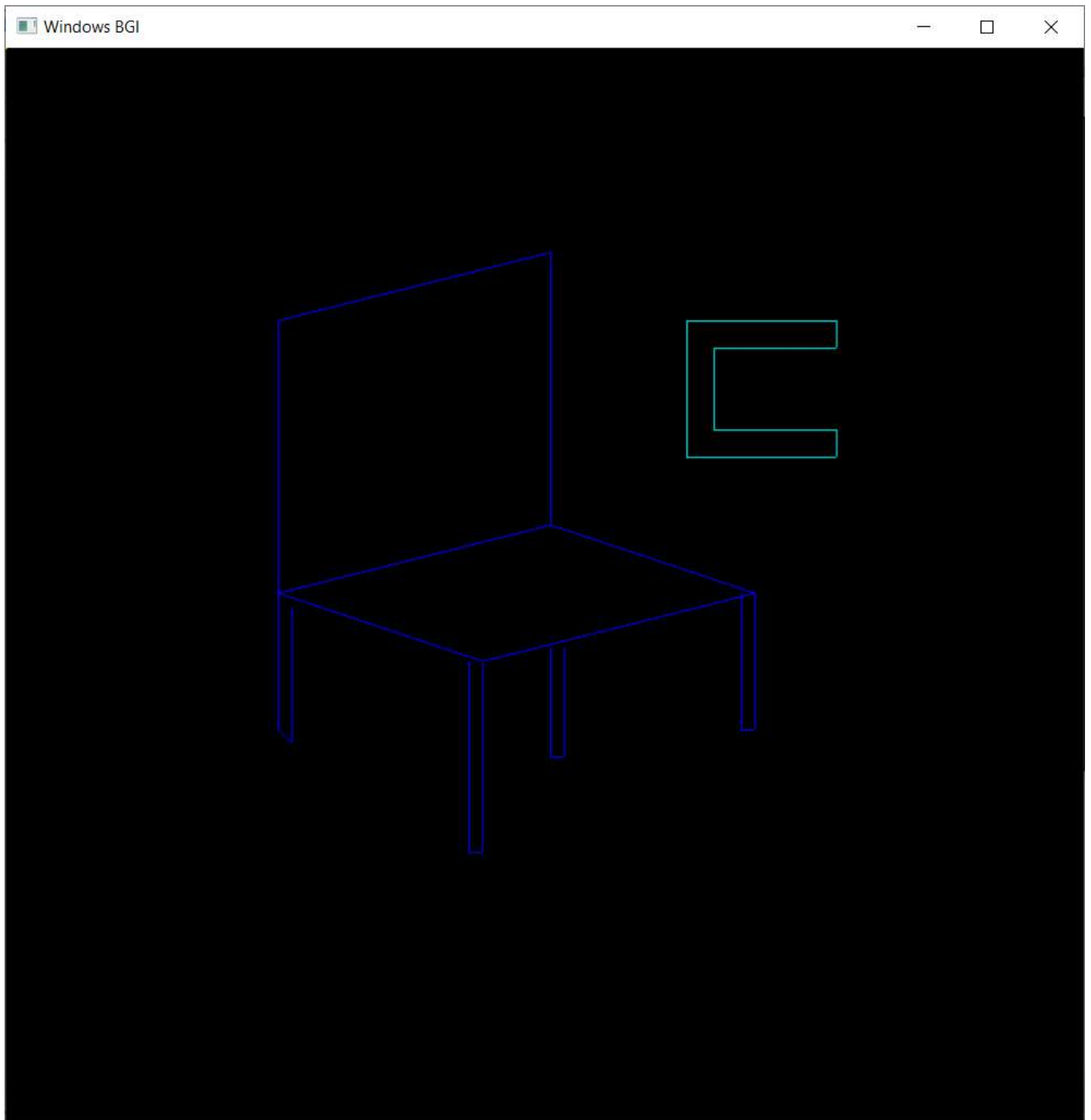
```
Dda_function(500, 200, 610, 200, 3);  
Dda_function(500, 300, 610, 300, 3);  
Dda_function(610, 280, 610, 300, 3);  
Dda_function(610, 200, 610, 220, 3);  
Dda_function(520, 220, 610, 220, 3);  
Dda_function(520, 280, 610, 280, 3);  
Dda_function(520, 220, 520, 280, 3);
```

```
getch();
```

```
closegraph();
```

```
}
```

OUTPUT



VIVA QUESTIONS:

Q1. Define `getpixel` and `putpixel` functions in `graphics.h`?

Ans. **getpixel():** Function `getpixel` returns the color of pixel present at point(x, y).

Declarations: `int getpixel(int x, int y);`

Putpixel(): Function `putpixel` plots a pixel at a point(x, y) of the specified color.

Declarations: `void putpixel(int x, int y, int color);`

Q2. What is the difference between DDA algorithm and Bresenham's Algorithm?

Ans.

DDA Algorithm	Bresenham's Line Algorithm
1. DDA Algorithm use floating point, i.e., Real Arithmetic.	1. Bresenham's Line Algorithm use fixed point, i.e., Integer Arithmetic
2. DDA Algorithms uses multiplication & division its operation	2. Bresenham's Line Algorithm uses only subtraction and addition its operation
3. DDA Algorithm is slowly than Bresenham's Line Algorithm in line drawing because it uses real arithmetic (Floating Point operation)	3. Bresenham's Algorithm is faster than DDA Algorithm in line because it involves only addition & subtraction in its calculation and uses only integer arithmetic.
4. DDA Algorithm is not accurate and efficient as Bresenham's Line Algorithm.	4. Bresenham's Line Algorithm is more accurate and efficient at DDA Algorithm.
5. DDA Algorithm can draw circle and curves but are not accurate as Bresenham's Line Algorithm	5. Bresenham's Line Algorithm can draw circle and curves with more accurate than DDA Algorithm.

Q3. Give disadvantages of DDA Algorithm?

Ans. It involves floating point additions rounding off is done. Accumulations of round off error cause accumulation of error.

Rounding off operations and floating point operations consumes a lot of time.

It is more suitable for generating line using the software. But it is less suited for hardware implementation.

Q4. Give disadvantages of Bresenham's algorithm?

Ans. This algorithm is meant for basic line drawing only Initializing is not a part of Bresenham's line algorithm. So to draw smooth lines, you should want to look into a different algorithm.



EXPERIMENT - 3

Computer Graphics and Multimedia

Bresenham's Algorithm

Write a C program to draw a line using Bresenham's algorithm.

Syeda Reeha Quasar

14114802719

Group - 3C7

EXPERIMENT – 3

AIM:

Write a C program to draw a line using Bresenham's 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.

Advantage:

1. It involves only integer arithmetic, so it is simple.
2. It avoids the generation of duplicate points.
3. It can be implemented using hardware because it does not use multiplication and division.
4. It is faster as compared to DDA (Digital Differential Analyzer) because it does not involve floating point calculations like DDA Algorithm.

Disadvantage:

1. This algorithm is meant for basic line drawing only. Initializing is not a part of Bresenham's line algorithm. So to draw smooth lines, you should want to look into a different algorithm.

Bresenham's Line Algorithm:

Step1: Start Algorithm

Step2: Declare variable $x_1, x_2, y_1, y_2, d, i_1, i_2, dx, dy$

Step3: Enter value of x_1, y_1, x_2, y_2

Where x_1, y_1 are coordinates of starting point
And x_2, y_2 are coordinates of Ending point

Step4: Calculate $dx = x_2 - x_1$

Calculate $dy = y_2 - y_1$

Calculate $i_1 = 2 * dy$

Calculate $i_2 = 2 * (dy - dx)$

Calculate $d = i_1 - dx$

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

If $dx < 0$

Then $x = x_2$

$y = y_2$

$x_{end} = x_1$

If $dx > 0$

Then $x = x_1$
 $y = y_1$
 $x_{end} = x_2$

Step6: Generate point at (x,y)coordinates.

Step7: Check if whole line is generated.

If $x \geq x_{end}$
Stop.

Step8: Calculate co-ordinates of the next pixel

If $d < 0$
Then $d = d + i_1$
If $d \geq 0$
Then $d = d + i_2$
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

Drawing Line using Bresenham's Algorithm

CODE

```
#include<stdio.h>
#include<graphics.h>
#include<conio.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;
        }
        x=x+1;
        p=p+2*dx;
    }
}
```

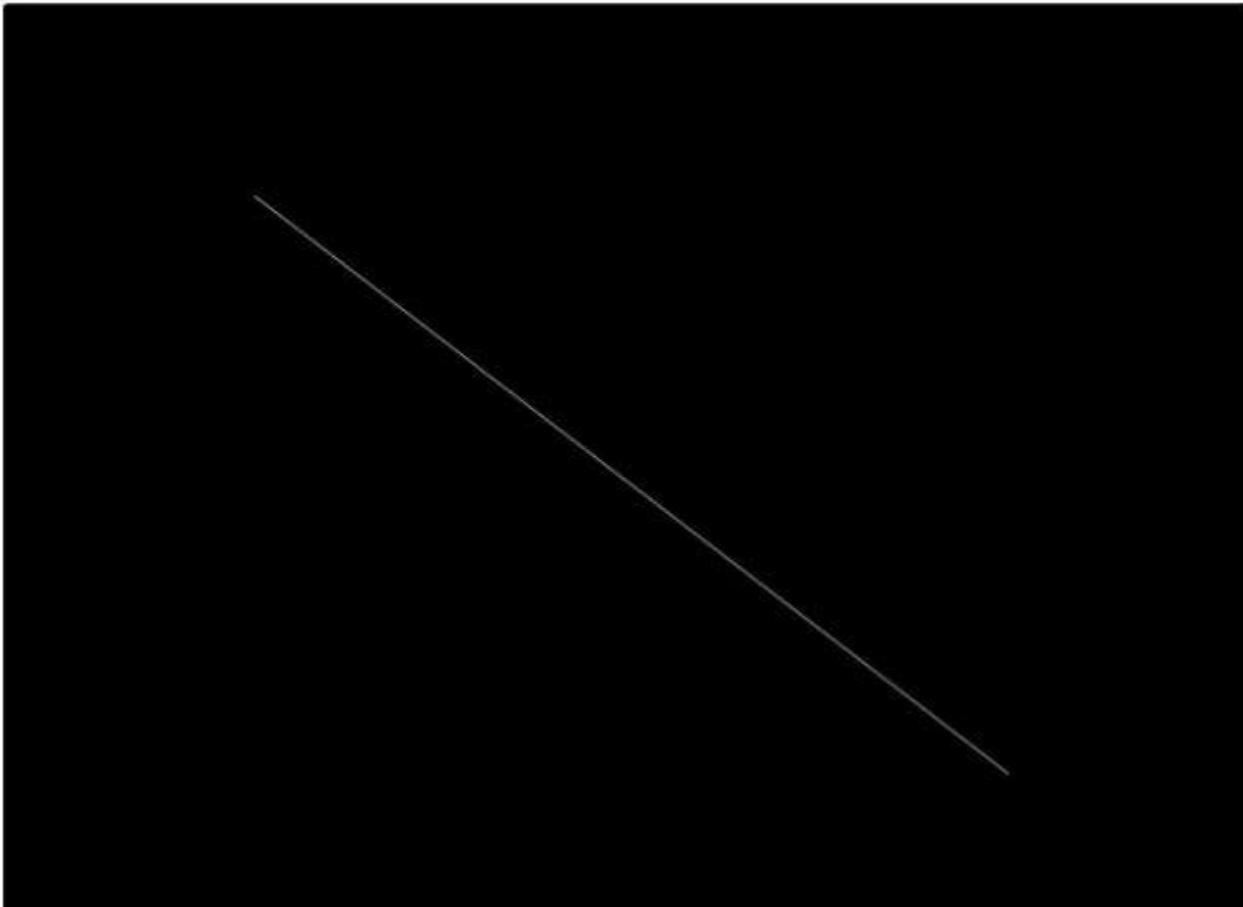
```

        p=p+2*dy-2*dx;
    }
    else
    {
        putpixel(x,y,7);
        p=p+2*dy;
    }
    x=x+1;
}
}

int main()
{
    initwindow(800,800);
    drawline(100,100,400,400);
    getch();
}

```

Windows BGI



Source Code:

```
#include <stdio.h>

#include <conio.h>

#include <graphics.h>

bresenham(int x0, int y0, int x1, int y1, int clr)

{

    float dx, dy, p, x, y, t, m;

    if ((x1 - x0) == 0)

    {

        m = y1 - y0;

    }

    else{

        m = (y1 - y0) / (x1 - x0);

    }

    if (abs(m) < 1)

    {

        if (x0 > x1)

        {

            t = x0;

            x0 = x1;

            x1 = t;
```

```

        t = y0;

        y0 = y1;

        y1 = t;

    }

    dx = abs(x1-x0);

    dy = abs(y1-y0);


    x = x0;

    y = y0;

    p = (2*dy)-dx;


    while(x<=x1)

    {

        putpixel(x, y, clr);

        x = x + 1;

        if (p >= 1)

        {

            if (m < 1)

            {

                y = y + 1;

            }

            else{

                y = y - 1;

            }

            p = p + (2 * dy) - (2 * dx);

        }

        else

```

```

        {
            p = p + (2*dy);
        }
    }
}

if (abs(m) > 1)
{
    if (y0 > y1)
    {
        t = x0;
        x0 = x1;
        x1 = t;

        t = y0;
        y0 = y1;
        y1 = t;
    }

    dx = abs(x1-x0);
    dy = abs(y1-y0);

    x = x0;
    y = y0;

    p = (2*dy)-dx;

    while(y<=y1)
    {

```

```

        putpixel(x, y, clr);

        y = y + 1;

        if (p >= 1)
        {
            if (m < 1)
            {
                x = x + 1;
            }

            else{
                x = x - 1;
            }

            p = p + (2 * dx) - (2 * dy);
        }

        else

        {

            p = p + (2*dx);

        }

    }

}

main()

{

    //pattern

    initwindow(800, 800);

    // diamond

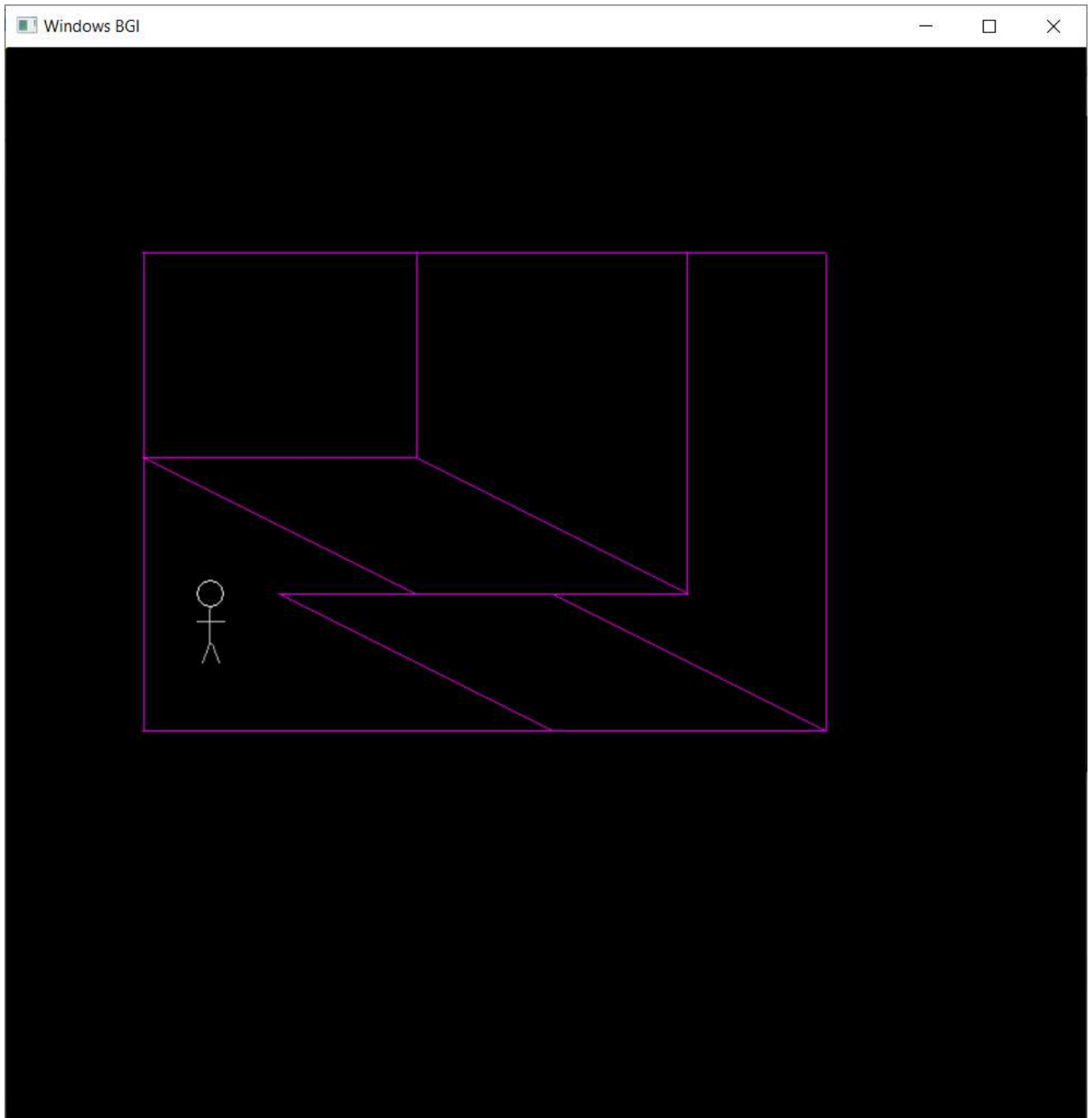
    bresenham(200, 400, 400, 400, 5);

    bresenham(200, 400, 400, 500, 5);

```

```
bresenham(400, 400, 600, 500, 5);  
bresenham(100, 500, 600, 500, 5);  
  
// diamond  
bresenham(100, 300, 300, 300, 5);  
bresenham(100, 300, 300, 400, 5);  
bresenham(300, 300, 500, 400, 5);  
bresenham(300, 400, 500, 400, 5);  
  
// joints  
bresenham(100, 150, 250, 150, 5);  
bresenham(100, 500, 100, 150, 5);  
bresenham(600, 500, 600, 150, 5);  
bresenham(400, 150, 600, 150, 5);  
bresenham(250, 150, 500, 150, 5);  
bresenham(500, 150, 500, 400, 5);  
  
bresenham(300, 300, 300, 150, 5);  
  
//man  
circle(150,400,10);  
bresenham(150,410,150,436, 7);  
bresenham(150,436,145,450, 7);  
bresenham(150,436,155,450, 7);  
bresenham(150,436,155,450, 7);  
bresenham(140,420,160,420, 7);  
bresenham(160,420,170,402, 7);  
getch();  
closegraph();}
```

OUTPUT



VIVA QUESTIONS:

Q1. What is the difference between DDA algorithm and Bresenham's Algorithm?

Ans.

DDA Algorithm	Bresenham's Line Algorithm
1. DDA Algorithm use floating point, i.e., Real Arithmetic.	1. Bresenham's Line Algorithm use fixed point, i.e., Integer Arithmetic
2. DDA Algorithms uses multiplication & division its operation	2. Bresenham's Line Algorithm uses only subtraction and addition its operation
3. DDA Algorithm is slowly than Bresenham's Line Algorithm in line drawing because it uses real arithmetic (Floating Point operation)	3. Bresenham's Algorithm is faster than DDA Algorithm in line because it involves only addition & subtraction in its calculation and uses only integer arithmetic.
4. DDA Algorithm is not accurate and efficient as Bresenham's Line Algorithm.	4. Bresenham's Line Algorithm is more accurate and efficient at DDA Algorithm.
5. DDA Algorithm can draw circle and curves but are not accurate as Bresenham's Line Algorithm	5. Bresenham's Line Algorithm can draw circle and curves with more accurate than DDA Algorithm.

Q2. What is `c:\tc\bgi`?

Ans. It specifies the directory path where `initgraph` looks for graphics drivers (*. BGI) first. If files are not there then `initgraph` will look for the current directory of your program.

Q3. Define `closegraph()`?

Ans. The header file `graphics.h` contains `closegraph()` function which closes the graphics mode, deallocates all memory allocated by graphics system and restores the screen to the mode it was in before you called `initgraph`.

Syntax : `void closegraph();`

Q4. What is `DETECT` graphics driver?

Ans. `DETECT` is an enumeration type that identifies the proper graphics driver. It tells the compiler that what graphics driver to use or to automatically detect the drive. If graphic driver is set to `DETECT`,

then initgraph sets graphic mode to the highest resolution available for the detected driver. Use -
intgd = DETECT, gm;

Q5. In Bresenham's line algorithm, if the distance $d1 < d2$ then decision parameter P_k is _____

- a) Positive**
- b) Equal**
- c) Negative**
- d) Option a or c**

Ans. C) Negative

Q6. The Algorithm which uses multiple processors to calculate pixel positions is

- a) Midpoint algorithm**
- b) Parallel line algorithm**
- c) Bresenham's line algorithm**
- d) All of the above mentioned**

Ans. b) Parallel line algorithm

Q3. Which algorithm is a faster method for calculating pixel position?

- a) Bresenham's line algorithm**
- b) Parallel line algorithm**
- c) Mid-point algorithm**
- d) DDA line algorithm**

Ans. d) DDA line algorithm



EXPERIMENT - 4

COMPUTER GRAPICS AND MULTIMEDIA

Midpoint Circle Generation Algorithm

To implement midpoint circle generation algorithm or bresenham's circle algorithm for drawing a circle of given center (x, y) and radius r

Syeda Reeha Quasar

14114802719

Group - 3C7

EXPERIMENT – 4

AIM:

To implement midpoint circle generation algorithm or bresenham's circle algorithm for drawing a circle of given center (x, y) and radius r.

THEORY:

Circles have the property of being highly symmetrical, which is handy when it comes to drawing them on a display screen.

- We know that there are 360 degrees in a circle. First, we see that a circle is symmetrical about the x axis, so only the first 180 degrees need to be calculated.
- Next, we see that it's also symmetrical about the y axis, so now we only need to calculate the first 90 degrees.
- Finally, we see that the circle is also symmetrical about the 45-degree diagonal axis, so we only need to calculate the first 45 degrees.
- We only need to calculate the values on the border of the circle in the first octant. The other values may be determined by symmetry

Bresenham's circle algorithm calculates the locations of the pixels in the first 45 degrees. It assumes that the circle is centered on the origin. So, for every pixel (x, y) it calculates, we draw a pixel in each of the eight octants of the circle. This is done till when the value of the y coordinate equals the x coordinate. The pixel positions for determining symmetry are given in the below algorithm.

- Assume that we have just plotted point (x_k, y_k)
- The next point is a choice between (x_k+1, y_k) and (x_k+1, y_k-1)
- We would like to choose the point that is nearest to the actual circle.
- So, we use decision parameter here to decide

Decision parameters:

1. In this the input radius r is there with a center (x_c, y_c). To obtain the first point on the circumference of a circle is centered on the origin as (x₀, y₀) = (0, r).
2. Calculate the initial decision parameters which are:
$$p_0 = 5/4 - r \text{ or } 1 - r$$
3. Now at each x_k position starting k=0, perform the following task.
if $p_k < 0$ then plotting point will be (x_k+1, y_k) and
$$p_{k+1} = p_k + 2(x_{k+1}) + 1$$

else the next point along the circle is (x_k+1, y_k-1) and
$$p_{k+1} = p_k + 2(x_{k+1}) + 1 - 2(y_{k+1})$$
4. Determine the symmetry points in the other quadrants.
5. Now move at each point by the given center that is:
$$x = x + x_c \quad y = y + y_c$$
6. At last repeat steps from 3 to 5 until the condition $x \geq y$

Symmetric pixel positions:

- `putpixel(xc+x,yc-y, GREEN); //For pixel(x,y)`
- `putpixel(xc+y,yc-x, GREEN); //For pixel(y,x)`
- `putpixel(xc+y,yc+x, GREEN); //For pixel(y,-x)`
- `putpixel(xc+x,yc+y, GREEN); //For pixel(x,-y)`
- `putpixel(xc-x,yc+y, GREEN); //For pixel(-x,-y)`
- `putpixel(xc-y,yc+x, GREEN); //For pixel(-y,-x)`
- `putpixel(xc-y,yc-x, GREEN); //For pixel(-y,x)`
- `putpixel(xc-x,yc-y, GREEN); //For pixel(-x,y)`

Midpoint Circle Generation Algorithm:

Given - Centre point of Circle = (X0 , Y0)

Radius of Circle = R

The points generation using Mid-Point Circle Drawing Algorithm involves the following steps-

Step-01:

Assign the starting point coordinates (X0, Y0) as –

$$X0 = 0$$

$$Y0 = R$$

Step-02:

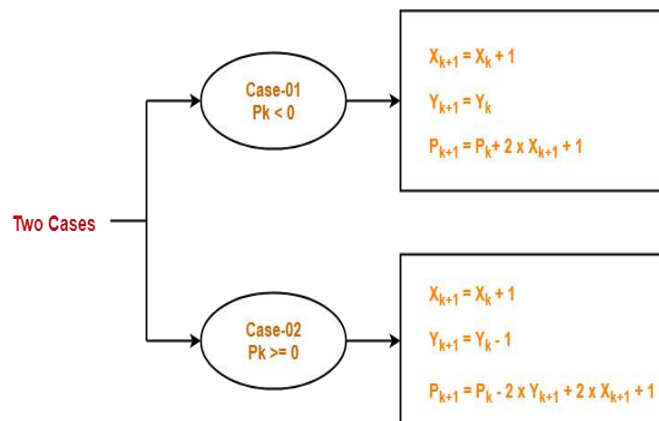
Calculate the value of initial decision parameter P0 as –

$$P0 = 1 - R$$

Step3:

Suppose the current point is (Xk , Yk) and the next point is (Xk+1, Yk+1).

Find the next point of the first octant depending on the value of decision parameter Pk.



Step4:

If the given centre point (X_0, Y_0) is not $(0, 0)$, then do the following and plot the point –

$$X_{plot} = X_c + X_0$$

$$Y_{plot} = Y_c + Y_0$$

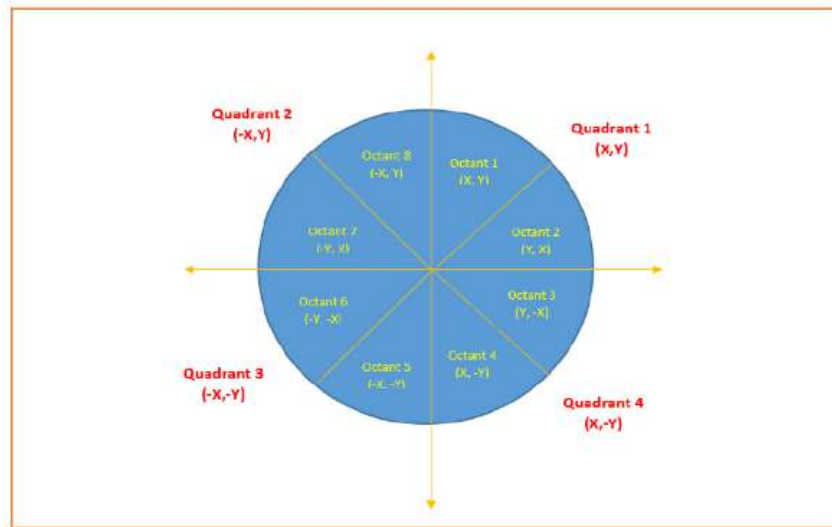
Here, (X_c, Y_c) denotes the current value of X and Y coordinates.

Step5:

Keep repeating Step-03 and Step-04 until $X_{plot} \geq Y_{plot}$.

Step6:

Step-05 generates all the points for one octant. To find the points for other seven octants, follow the eight symmetry property of circle



Step7: End Algorithm

Algorithm implementation:

```
#include<graphics.h>
```

```
#include<conio.h>
```

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

```

main(){

    initwindow(800,800);

    int i, r=100, x=0,y,xc=200,yc=200;

    float d;

    d=1.25-r;

    y=r;

    do{

        if(d<0.0){

            x=x+1;

            d=d+2*x+1;

        }else{

            x=x+1;

            y=y-1;

            d=d+2*x-2*y+10;

        }

        putpixel(xc+x,yc+y,7);

        putpixel(xc-y,yc-x,7);

        putpixel(xc+y,yc-x,7);

        putpixel(xc-y,yc+x,7);

        putpixel(xc+y,yc+x,7);

        putpixel(xc-x,yc-y,7);

        putpixel(xc+x,yc-y,7);

        putpixel(xc-x,yc+y,7);

    }

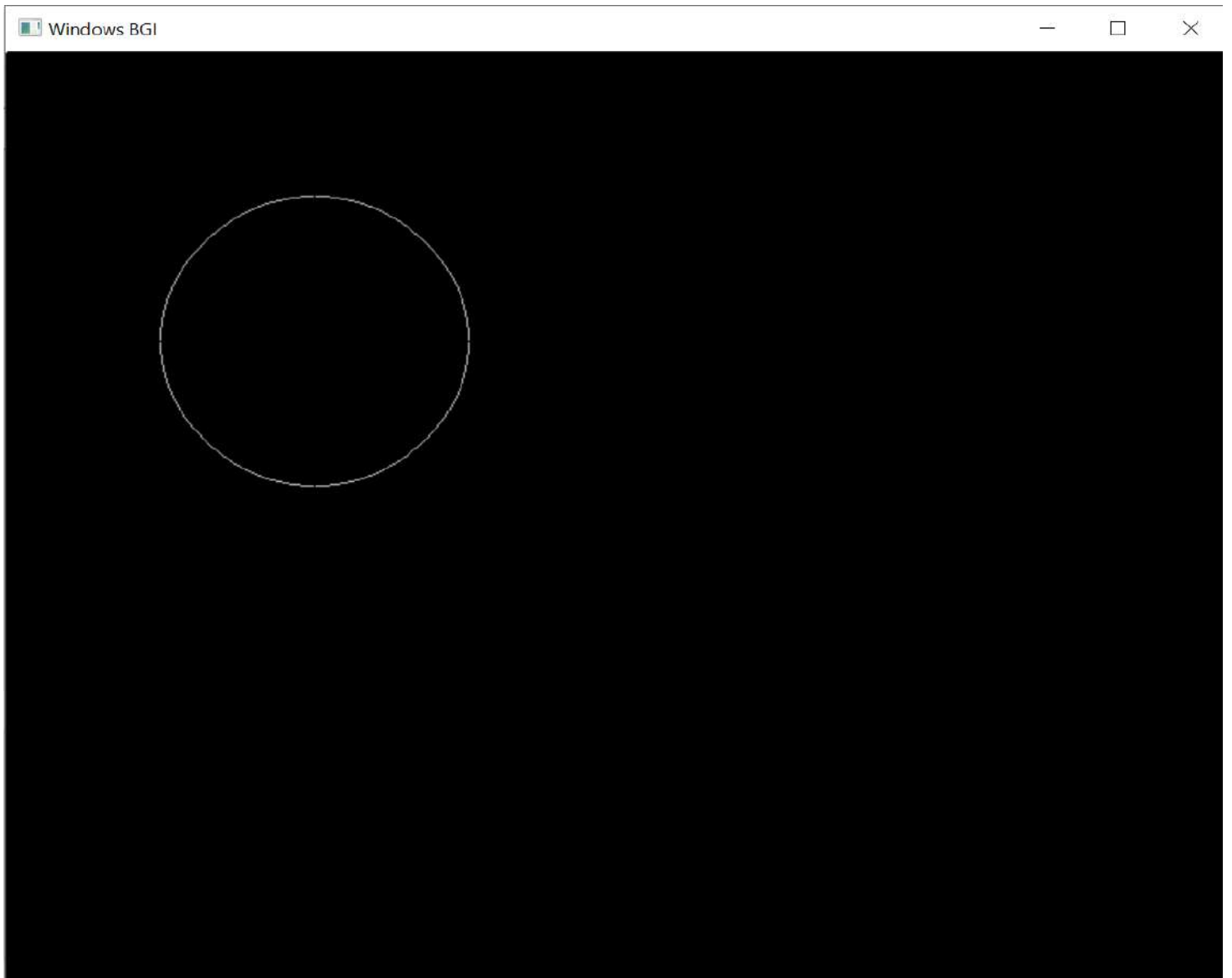
    while(x<y);

    getch();

}

```

OUTPUT:



Source Code:

```
#include<stdio.h>

#include<conio.h>

#include<graphics.h>

circle_func(int xc,int yc,int r, int clr )

{

    int i,y,x=0;

    float d;

    d=1.25-r;

    y=r;

    do{

        if(d<0.0){

            x=x+1;

            d=d+2*x+1;

        }

        else{

            x=x+1;

            y=y-1;

            d=d+2*x-2*y+1;

        }

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

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

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

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

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

        putpixel(xc-x,yc-y,clr);
```

```
    putpixel(xc+x,yc-y,clr);  
    putpixel(xc-x,yc+y,clr);  
}  
while(x<y);  
}
```

```
main()
```

```
{  
    initwindow(800,800);  
    int i = 100;  
    while (i <= 700){  
        circle_func(i,200,10, 1);  
        circle_func(i,200,100, 7);  
        circle_func(i,180,50, 2);  
        circle_func(i,270,100, 3);  
        i += 100;  
        delay (100);  
}
```

```
i = 200;
```

```
while (i < 700){  
    circle_func(i,400,10, 1);  
    circle_func(i,400,100, 7);  
    circle_func(i,380,50, 2);  
    circle_func(i,370,100, 3);  
    i += 100;  
    delay (100);
```

```
}
```

```
    for (i = 1; i <= 100; i++)
```

```
{
```

```
    circle_func(400,500,i, 4);
```

```
}
```

```
delay (100);
```

```
for (i = 1; i <= 50; i++)
```

```
{
```

```
    circle_func(400,300,i, 1);
```

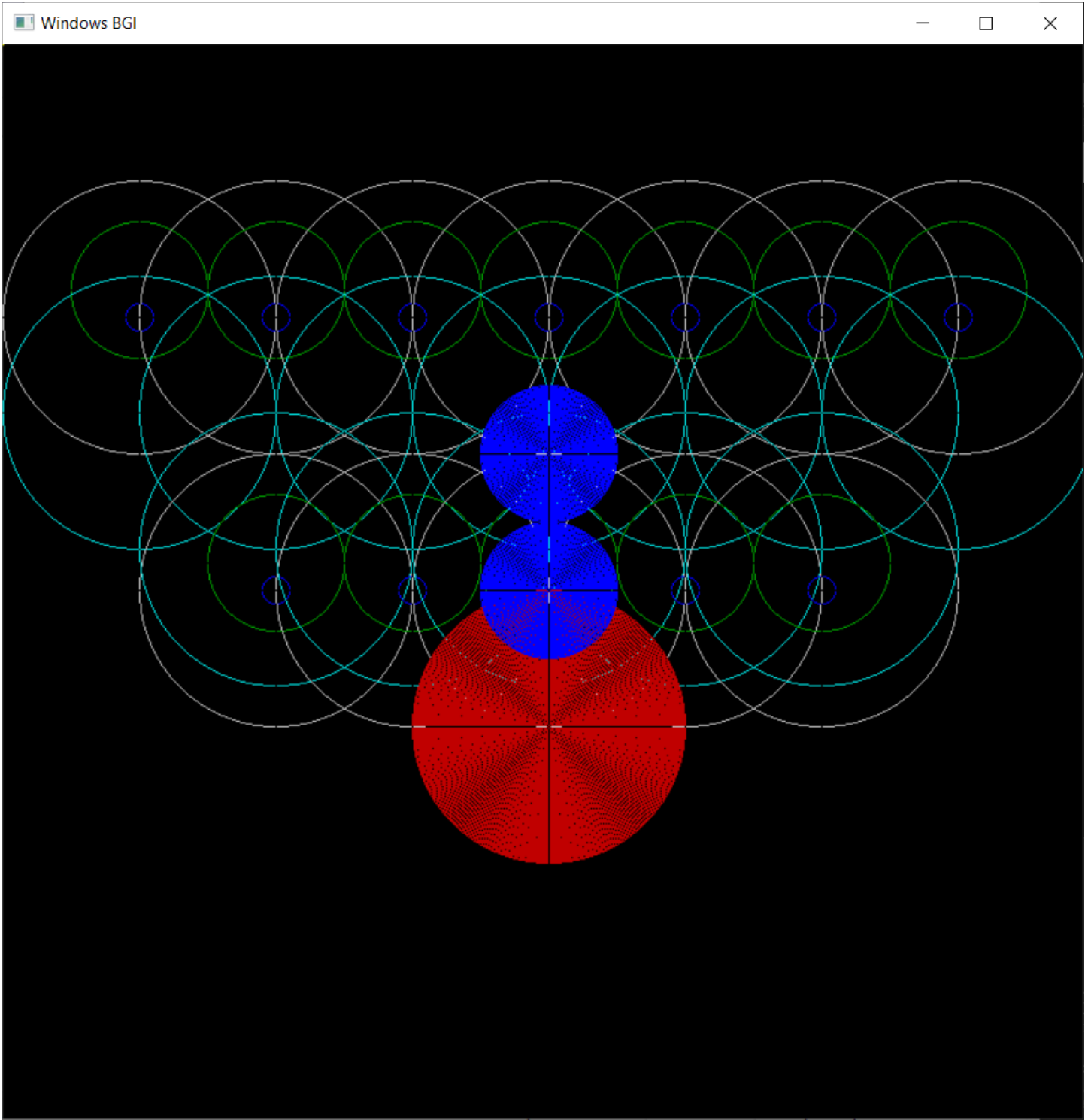
```
    circle_func(400,400,i, 1);
```

```
}
```

```
getch();
```

```
}
```

OUTPUT:



VIVA QUESTIONS:

Q1. How many parameters are required to draw the circle?

Ans.

3 Parameters are required to draw a circle that are – coordinates of the center x, y and the radius of the circle.

Syntax : circle(x, y, radius); where, (x, y) is center of the circle. 'radius' is the Radius of the circle.

Q2. Why we calculate the initial value of the decision parameter as $P_0 = 5/4 - r \approx 1 - r$?

Ans. $P_0 = 5/4 - r \approx 1 - r$

For initial point, $(x_0, y_0) = (0, r)$

$$\begin{aligned} P_0 &= f_{\text{circle}}(1, r - 1/2) \\ &= 1 + (r - 1/2)^2 - r^2 \\ &= 5/4 - r \\ &\sim 1 - r \end{aligned}$$

Q3. What are symmetry points?

Ans.

Point Symmetry is when every part has a matching part: the same distance from the central point, but in the opposite direction. A circle have infinite points of symmetry. Circle drawing algorithms take the advantage of 8 symmetry property of circle. Every circle has 8 octants and the circle drawing algorithm generates all the points for one octant. The points for other 7 octants are generated by changing the sign towards X and Y coordinates. implementation.

Q4. What is plot points?

Ans.

A plot is a graphical technique for representing a data set, usually as a graph showing the relationship between two or more variables and the points we plot are called the plot points. Point plotting is an elementary mathematical skill required in analytic geometry. Plot points vary as per the conditions given. In the mid-point algorithm we calculate the P_k and according to that we decide the next plotting points.

Q5. What is circle midpoint?

Ans.

The midpoint circle algorithm is an algorithm used to determine the points needed for rasterizing a circle. Bresenham's circle algorithm is derived from the midpoint circle algorithm. This is an algorithm which is used to calculate the entire perimeter points of a circle in a first octant so that the points of the other octant can be taken easily as they are mirror points; this is due to circle property as it is symmetric about its center. In this technique algorithm determines the midpoint between the next 2 possible consecutive pixels and then checks whether the midpoint is inside or outside the circle and illuminates the pixel accordingly.



EXPERIMENT - 5

COMPUTER GRAPICS AND MULTIMEDIA

BRESENHAM'S CIRCLE DRAWING ALGORITHM

Implementation of Bresenham's circle drawing algorithm

Syeda Reeha Quasar

14114802719

Group - 3C7

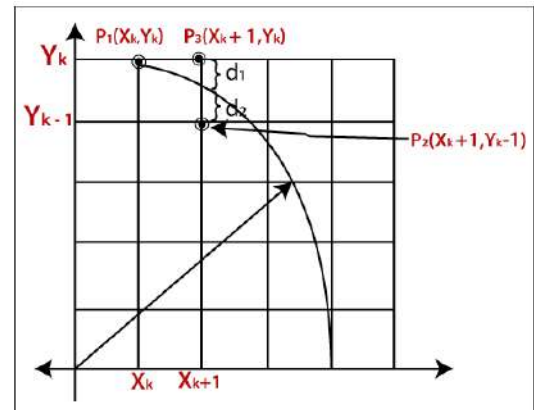
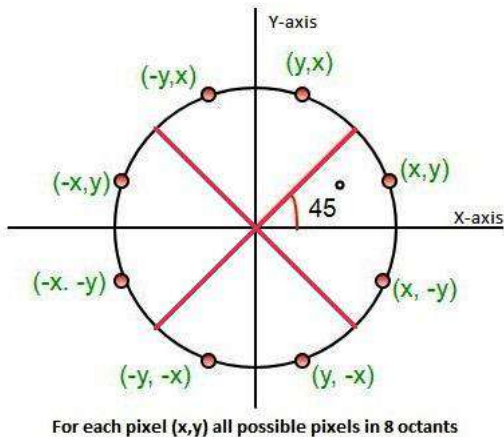
EXPERIMENT – 5

AIM:

Implementation of Bresenham's circle drawing algorithm

THEORY:

- Bresenham's circle drawing algorithm selects the nearest pixel position to complete the arc.
- It uses only integer arithmetic which makes it faster than any other algorithms.
- The circle can be divided into 8 octants each of 45 degree where centre of the circle is (0,0)



Bresenham Circle Algorithm:

Pseudocode:

Step 1: Set initial values of (xc, yc) and (x, y).

Where (xc,yc) are center of the circle

(x,y) are the starting point.

x= 0 and y= r (r is the radius of the circle)

Step 2: Calculate decision parameter d such that $d = 3 - (2 * r)$.

Step 3: Call the display_Circle(int xc, int yc, int x, int y) function to display initial point (0,r).

Step 4: Repeat steps 5 to 8 until $(x < = y)$

Step 5: Increment value of x.

Step 6: If $d < 0$, set $d = d + (4 \cdot x) + 6$

Step 7: Else, set $d = d + 4 \cdot (x - y) + 10$ and decrement y by 1.

Step 8: call `display_Circle(int xc, int yc, int x, int y)` method.

Step 9: Exit.

Algorithm implementation:

Source Code:

```
#include <graphics.h>
```

```
#include <stdlib.h>
```

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

```
#include <conio.h>
```

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

```
void EightSymmetricPlot(int xc, int yc, int x, int y, int clr)
```

```
{  
    putpixel(x + xc, y + yc, clr);  
    putpixel(x + xc, -y + yc, clr);  
    putpixel(-x + xc, -y + yc, clr);  
    putpixel(-x + xc, y + yc, clr);  
    putpixel(y + xc, x + yc, clr);  
    putpixel(y + xc, -x + yc, clr);  
    putpixel(-y + xc, -x + yc, clr);  
    putpixel(-y + xc, x + yc, clr);  
}
```

```
void BresenhamCircle(int xc, int yc, int r, int clr)
```

```

{
    int x = 0, y = r, d = 3 - ( 2 * r);
    EightSymmetricPlot(xc, yc, x, y, clr);

    while (x <= y)
    {
        if (d <= 0)
        {
            d = d + (4 * x) + 6;
        }
        else
        {
            d = d + (4 * x) - (4 * y) + 10;
            y = y - 1;
        }
        x = x + 1;
        EightSymmetricPlot(xc, yc, x, y, clr);
        delay(100);
    }
}

```

```

int main(void)
{
    int xc, yc, r, c = 0, i, j, gdriver=DETECT, gmode;
    initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

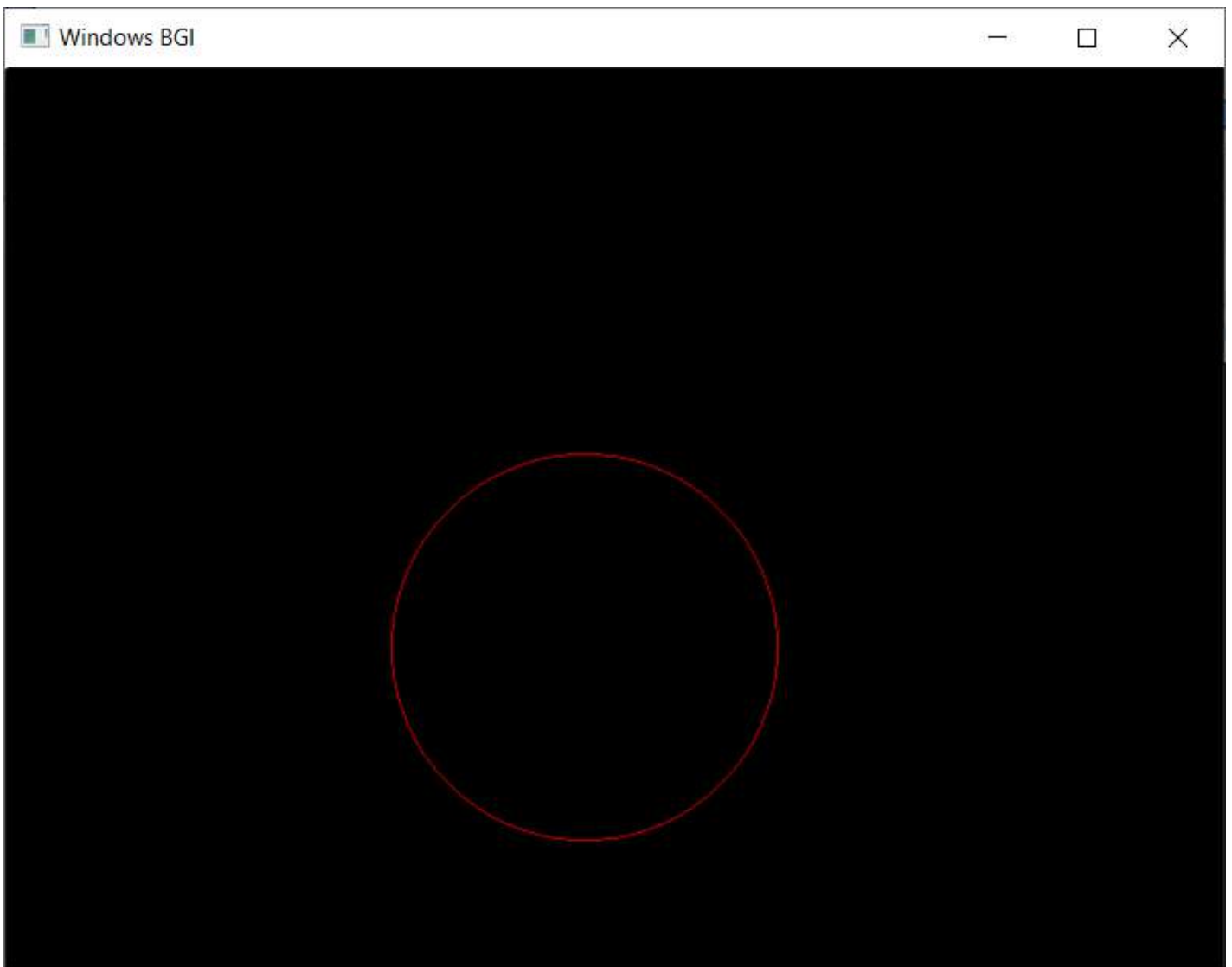
    printf("\n Enter the values of xc and yc: \n");
    scanf("%d %d", &xc, &yc);
    printf("\n Enter the value of radius: \n");
    scanf("%d", &r);

    BresenhamCircle(xc, yc, r, 4);
}

```

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

OUTPUT:



Pattern and design using the algorithm:

SOURCE CODE:

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

void EightSymmetricPlot(int xc, int yc, int x, int y, int clr)
{
    putpixel(x + xc, y + yc, clr);
    putpixel(x + xc, -y + yc, clr);
    putpixel(-x + xc, -y + yc, clr);
    putpixel(-x + xc, y + yc, clr);
    putpixel(y + xc, x + yc, clr);
    putpixel(y + xc, -x + yc, clr);
    putpixel(-y + xc, -x + yc, clr);
    putpixel(-y + xc, x + yc, clr);
}

void BresenhamCircle(int xc, int yc, int r, int clr)
{
    int x = 0, y = r, d = 3 - (2 * r);

    EightSymmetricPlot(xc, yc, x, y, clr);

    while (x <= y)
    {
        if (d <= 0)
        {
            d = d + (4 * x) + 6;
        }
    }
}
```

```

        else
        {
            d = d + (4 * x) - (4 * y) + 10;
            y = y - 1;
        }
        x = x + 1;
        EightSymmetricPlot(xc, yc, x, y, clr);
//        delay(100);
    }
}

```

```

int main(void)
{
    int xc, yc, r, c = 0, i, j, gdriver=DETECT, gmode;
    initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

    for (i = 10; i < 100; i++)
    {
        BresenhamCircle(300, 300, i, 2);
    }
    BresenhamCircle(300, 300, 110, 2);
    for (i = 10; i < 50; i++)
    {
        BresenhamCircle(100, 100, i, 1);
    }
    BresenhamCircle(100, 100, 60, 1);
    for (i = 10; i < 70; i++)
    {
        BresenhamCircle(200, 200, i, 4);
    }
    BresenhamCircle(200, 200, 80, 4);
}

```

```

    for (i = 10; i < 60; i++)
    {
        BresenhamCircle(400, 400, i, 5);
    }

    BresenhamCircle(400, 400, 70, 5);

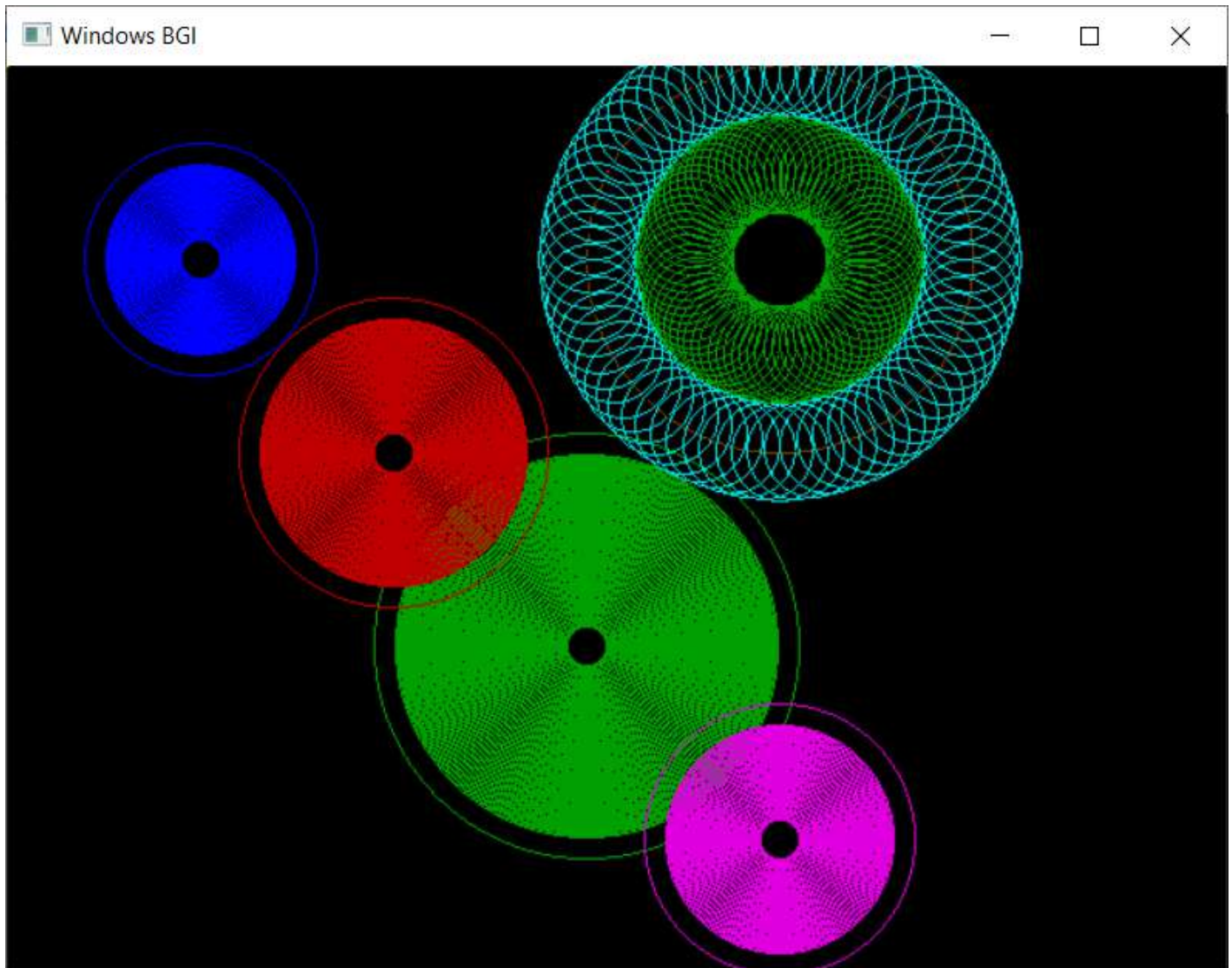
    BresenhamCircle(400, 100, 100, 6);

    //rangoli pattern circle
    int x, y, color, angle = 0;
    struct arccoordstype a, b;
    while (angle <= 360)
    {
        setcolor(BLACK);
        arc(400, 100, angle, angle + 2, 50);
        setcolor(RED);
        getarccoords(&a);
        BresenhamCircle(a.xstart, a.ystart, 25, 2);
        setcolor(BLACK);
        arc(400, 100, angle, angle + 2, 100);
        getarccoords(&a);
        setcolor(GREEN);
        BresenhamCircle(a.xstart, a.ystart, 25, 3);
        angle = angle+5;
        delay(50);
    }

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

```

OUTPUT:



VIVA QUESTIONS:

Q1. Explain the concept of 8-way symmetry?

Ans.

Circle is an eight-way symmetric figure. The shape of circle is the same in all quadrants. In each quadrant, there are two octants. If the calculation of the point of one octant is done, then the other seven points can be calculated easily by using the concept of eight-way symmetry.

Any circle follows 8-way symmetry. This means that for every point (x, y) 8 points can be plotted. These (x, y) , (y, x) , $(-y, x)$, $(-x, y)$, $(-x, -y)$, $(-y, -x)$, $(y, -x)$, $(x, -y)$. For any point $(x + a, y + b)$, points $(x \pm a, y \pm b)$ and $(y \pm a, x \pm b)$ also lie on the same circle.

Q2. How is the nearest pixel position calculated to complete the arc using bresenham's algorithm?

Ans.

In Bresenham's algorithm at any point (x, y) we have two option either to choose the next pixel i.e. $(x+1, y)$; $(x+1, y-1)$. And this can be decided by using the decision parameter d as:

- If $d > 0$, then $(x+1, y-1)$ is to be chosen as the next pixel as it will be closer to the arc.
- else $(x+1, y)$ is to be chosen as next pixel.

Q3. Bresenham's algorithm uses which type of arithmetic to find the pixel position?

Ans.

Bresenham's algorithm implemented entirely with integer numbers and the integer arithmetic. It only uses addition and subtraction and avoids heavy operations like multiplication and division.

Q4. How is the bresenham's circle drawing algorithm different from mid-point circle drawing algorithm?

Ans.

Bresenham's line algorithm determines the points of a raster that have to be selected to form an approximate straight line between two points. It is used in a bitmap image to draw line primitives. It uses integer addition, bit shifting and subtraction which are cheap operations in computer architectures. It is one of the earliest algorithms and is important even in the present because of its speed and simplicity. Its extension can be used for drawing circles.

Midpoint circle algorithm is used in computer graphics to determine the points required for rasterizing a circle. The Bresenham's circle algorithm is derived from it. The midpoint circle algorithm may be generalized to conic sections.



EXPERIMENT 6

COMPUTER GRAPICS AND MULTIMEDIA

Aim

Write C Programs for the implementation of 2D transformations.

Syeda Reeha Quasar
14114802719
3C7

EXPERIMENT 6

AIM:

Write C Programs for the implementation of 2D transformations.

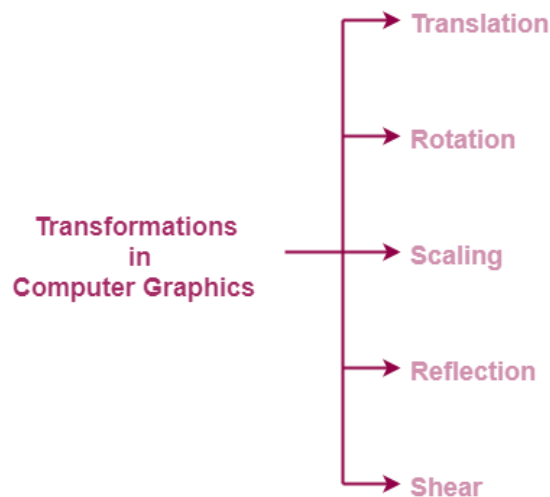
THEORY:

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.

In computer graphics, various transformation techniques are-

1. Translation
2. Rotation
3. Scaling
4. Reflection
5. Shear



In this experiment, we will discuss about 2D Translation, Rotation and Scaling in Computer Graphics.

2D Translation in Computer Graphics-

In Computer graphics,

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

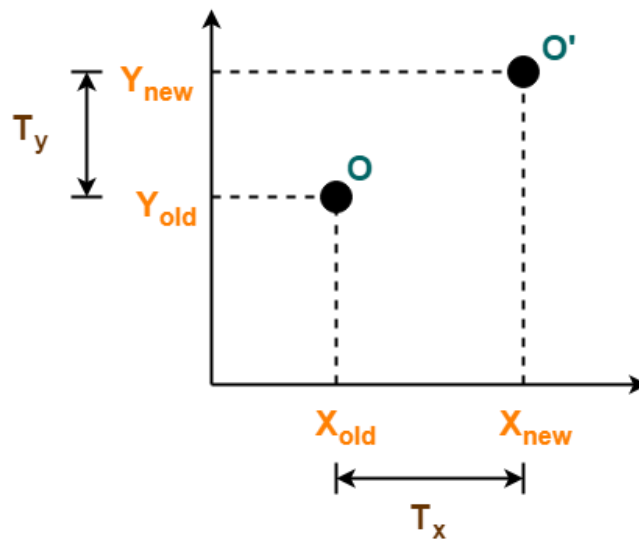
Consider a point object O has to be moved from one position to another in a 2D plane.

Let-

- Initial coordinates of the object $O = (X_{old}, Y_{old})$
- New coordinates of the object O after translation = (X_{new}, Y_{new})
- Translation vector or Shift vector = (T_x, T_y)

Given a Translation vector (T_x, T_y) -

- T_x defines the distance the X_{old} coordinate has to be moved.
- T_y defines the distance the Y_{old} coordinate has to be moved.



2D Translation in Computer Graphics

This translation is achieved by adding the translation coordinates to the old coordinates of the object as-

- $X_{new} = X_{old} + T_x$ (This denotes translation towards X axis)
- $Y_{new} = Y_{old} + T_y$ (This denotes translation towards Y axis)

In Matrix form, the above translation equations may be represented as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

Translation Matrix

- The homogeneous coordinates representation of (X, Y) is (X, Y, 1).
- Through this representation, all the transformations can be performed using matrix / vector multiplications.

The above translation matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \\ 1 \end{bmatrix}$$

Translation Matrix
(Homogeneous Coordinates Representation)

2D Rotation in Computer Graphics-

In Computer graphics,

2D Rotation is a process of rotating an object with respect to an angle in a two dimensional plane.

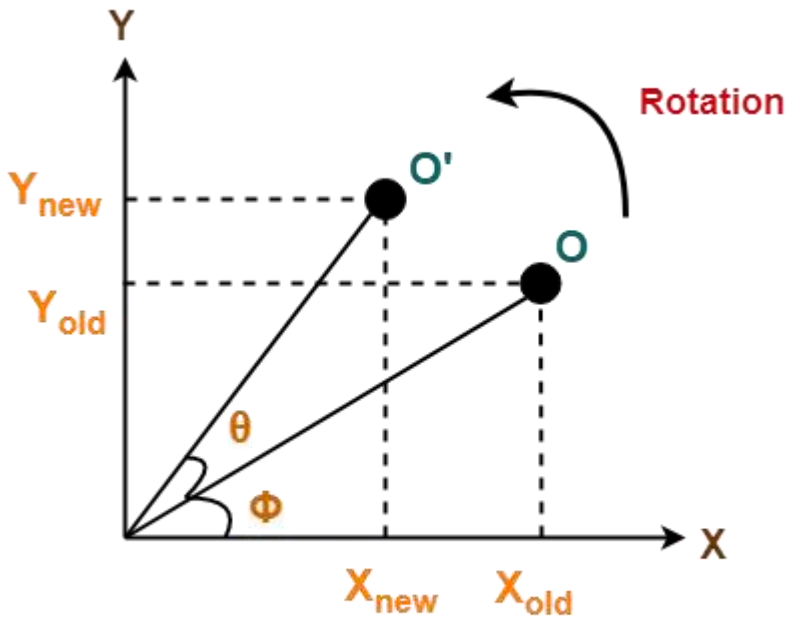
one angle to another in a 2D plane.

Consider a point object O has to be rotated from

Let-

- Initial coordinates of the object O = (X_{old}, Y_{old})
- Initial angle of the object O with respect to origin = Φ
- Rotation angle = θ

- New coordinates of the object O after rotation = $(X_{\text{new}}, Y_{\text{new}})$



2D Rotation in Computer Graphics

This rotation is achieved by using the following rotation equations-

- $X_{\text{new}} = X_{\text{old}} \times \cos\theta - Y_{\text{old}} \times \sin\theta$
- $Y_{\text{new}} = X_{\text{old}} \times \sin\theta + Y_{\text{old}} \times \cos\theta$

In Matrix form, the above rotation equations may be represented as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \end{bmatrix}$$

Rotation Matrix

For homogeneous coordinates, the above rotation matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \\ 1 \end{bmatrix}$$

Rotation Matrix
(Homogeneous Coordinates Representation)

2D Scaling in Computer Graphics-

In computer graphics, scaling is a process of modifying or altering the size of objects.

- Scaling may be used to increase or reduce the size of object.
- Scaling subjects the coordinate points of the original object to change.
- Scaling factor determines whether the object size is to be increased or reduced.
- If scaling factor > 1 , then the object size is increased.
- If scaling factor < 1 , then the object size is reduced.

Consider a point object O has to be scaled in a 2D plane.

Let-

- Initial coordinates of the object $O = (X_{\text{old}}, Y_{\text{old}})$
- Scaling factor for X-axis = S_x
- Scaling factor for Y-axis = S_y
- New coordinates of the object O after scaling = $(X_{\text{new}}, Y_{\text{new}})$

This scaling is achieved by using the following scaling equations-

- $X_{\text{new}} = X_{\text{old}} \times S_x$
- $Y_{\text{new}} = Y_{\text{old}} \times S_y$

In Matrix form, the above scaling equations may be represented as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \end{bmatrix}$$

Scaling Matrix

For homogeneous coordinates, the above scaling matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \\ 1 \end{bmatrix}$$

Scaling Matrix
(Homogeneous Coordinates Representation)

SOURCE CODE:

```
#include <stdio.h>

#include <graphics.h>

#include <conio.h>

#include <math.h>


void smiley(int x, int y, int r)

{

    setcolor(YELLOW);


    // creating circle and fill it with

    // yellow color using floodfill.

    circle(x, y, r);

    setfillstyle(SOLID_FILL, YELLOW);

    floodfill(x, y, YELLOW);


    // Set color of background to black

    setcolor(BLACK);

    setfillstyle(SOLID_FILL, BLACK);


    // Use fill ellipse for creating eyes

    fillellipse(x + 10, y - 15, 2, 6);
```



```
fillellipse(x - 10, y - 15, 2, 6);
```

```
// Use ellipse for creating mouth
```

```
ellipse(x, y, y + 105, x + 35, r - 20, r - 31);
```

```
ellipse(x, y, y + 105, x + 35, r - 20, r - 30);
```

```
ellipse(x, y, y + 105, x + 35, r - 20, r - 29);
```

```
}
```

```
int rotation(int x, int y, int r, int flag)
```

```
{
```

```
    int p, q;
```

```
    p = abs(x * cos(r) - y * sin(r));
```

```
    q = abs(y * cos(r) + x * sin(r));
```

```
    if (flag == 1)
```

```
        return p;
```

```
    return q;
```

```
}
```

```
void rotate(float angle){
```

```
int x1,x2,x3,x4;
```

```
int y1,y2,y3,y4;
```

```
int refx,refy;
```

```
int ax1,ax2,ax3,ax4,ay1,ay2,ay3,ay4;

angle=angle*(3.14/180);

refx=100;

refy=100;

x1=100;

y1=100;

x2=150;

y2=100;

x3=150;

y3=150;

x4=100;

y4=150;

ax1=refy+(x1-refx)*cos(angle)-(y1-refy)*sin(angle);

ay1=refy+(x1-refx)*sin(angle)+(y1-refy)*cos(angle);

ax2=refy+(x2-refx)*cos(angle)-(y2-refy)*sin(angle);

ay2=refy+(x2-refx)*sin(angle)+(y2-refy)*cos(angle);

ax3=refy+(x3-refx)*cos(angle)-(y3-refy)*sin(angle);

ay3=refy+(x3-refx)*sin(angle)+(y3-refy)*cos(angle);

ax4=refy+(x4-refx)*cos(angle)-(y4-refy)*sin(angle);

ay4=refy+(x4-refx)*sin(angle)+(y4-refy)*cos(angle);

setcolor(4);

rectangle(100,150,150,100);
```

```
line(ax1,ay1,ax2,ay2);
```

```
line(ax2,ay2,ax3,ay3);
```

```
line(ax3,ay3,ax4,ay4);
```

```
line(ax4,ay4,ax1,ay1);
```

```
}
```

```
main()
```

```
{    initwindow(800, 800);
```

```
    rectangle(150, 200, 300, 400);
```

```
    circle(200, 200, 100);
```

```
    smiley(300, 100, 100);
```

```
    // translation
```

```
    printf("\n TRANSLATION \n\n");
```

```
    printf("Enter the value you want the x-coordinate to: \n");
```

```
    int x, y;
```

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

```
    printf("Enter the value you want the y-coordinate to: \n");
```

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

```
    rectangle(150 + x, 200 + y, 300 + x, 400 + y);
```

```
    circle(200 + x, 200 + y, 100);
```

```
    smiley(300 + x, 100 + y, 100);
```

```

// scaling

printf("\n\n scaling \n\n");

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

int d;

scanf("%d", &d);

rectangle(150 * d , 200 * d, 300 * d, 400 * d);

circle(200, 200 , 100 * d);


// rotation

printf("\n\n ROTATION \n\n");

printf("Enter the angle\n");

int r;

scanf("%d", &r);

// 2 ways to print:

// rectangle(rotation(15, 20, r, 1), rotation(15, 20, r, 0), rotation(30, 40, r, 1), rotation(30, 40, r, 0));

rotate(r);

getch();

closegraph();

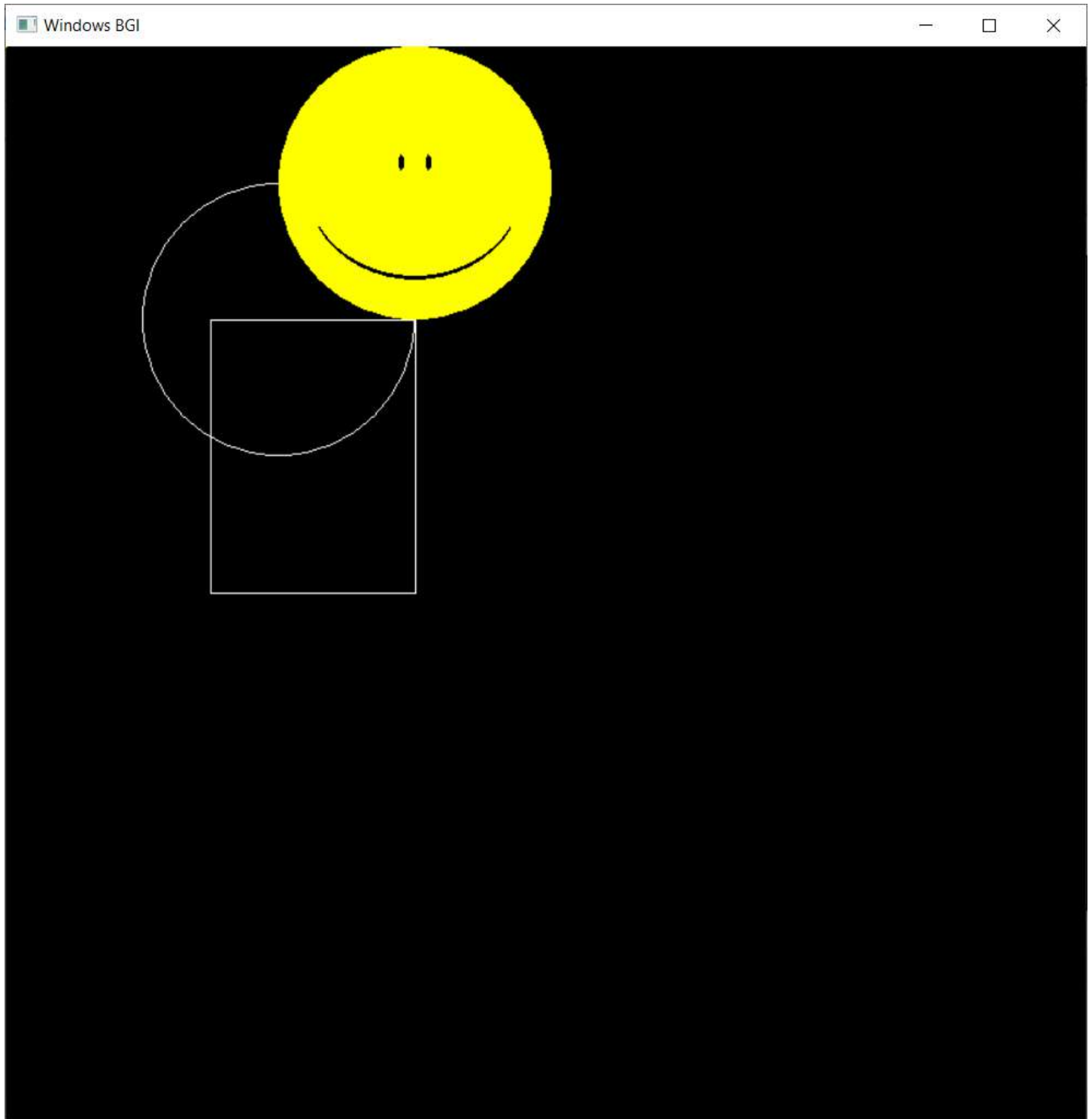
}

```

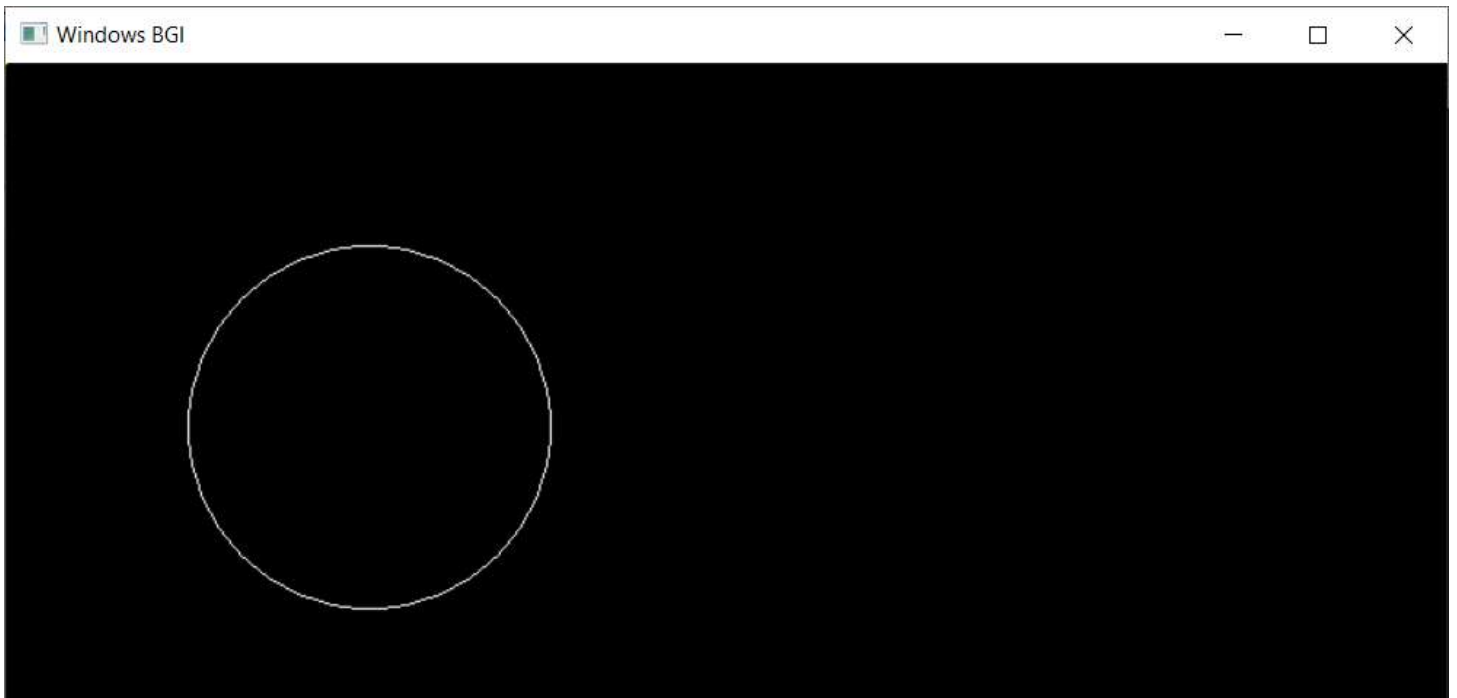
OUTPUT:

SHAPES GOING TO USE:

- Square
- Circle



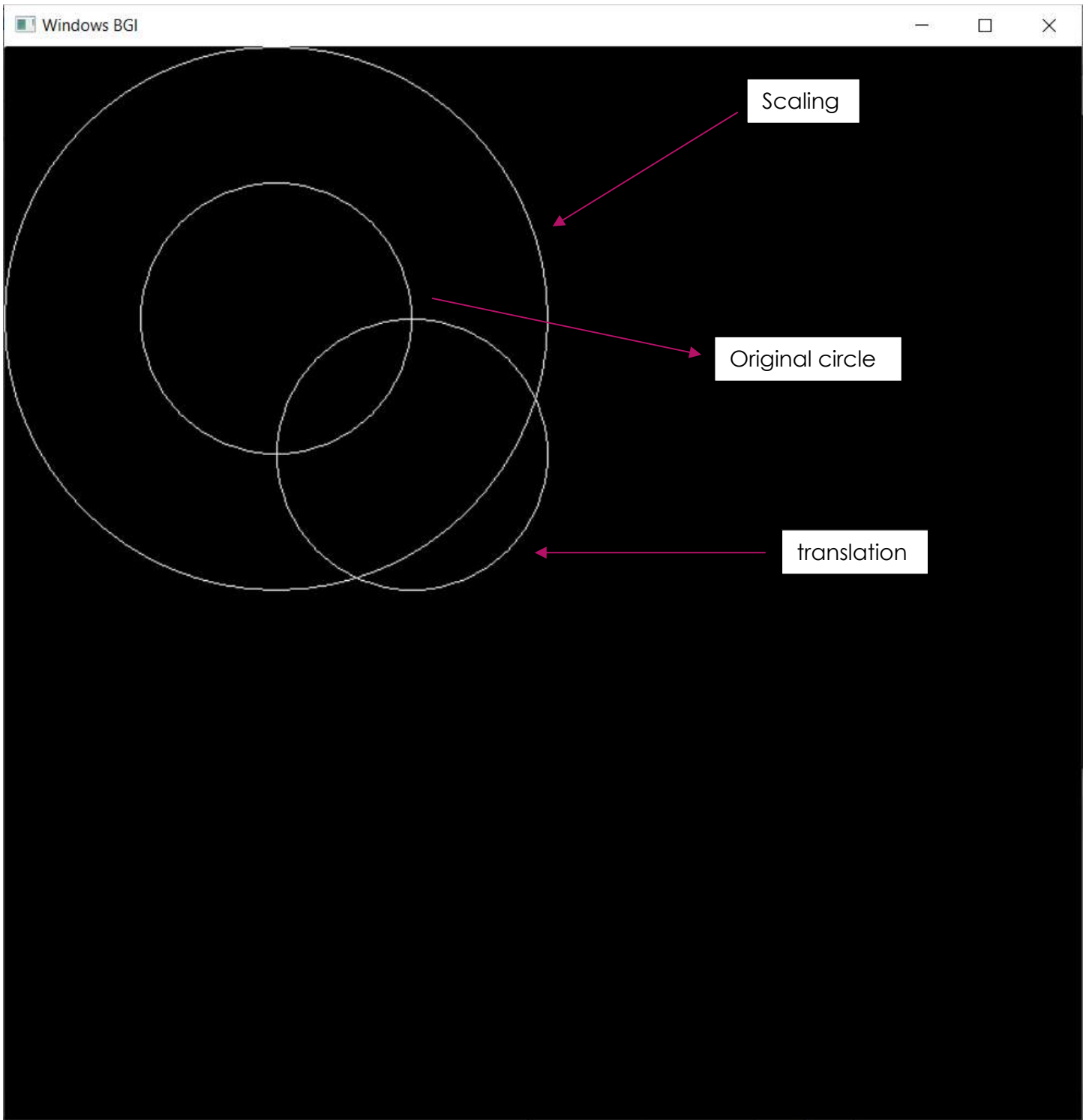
All transformations on circle:



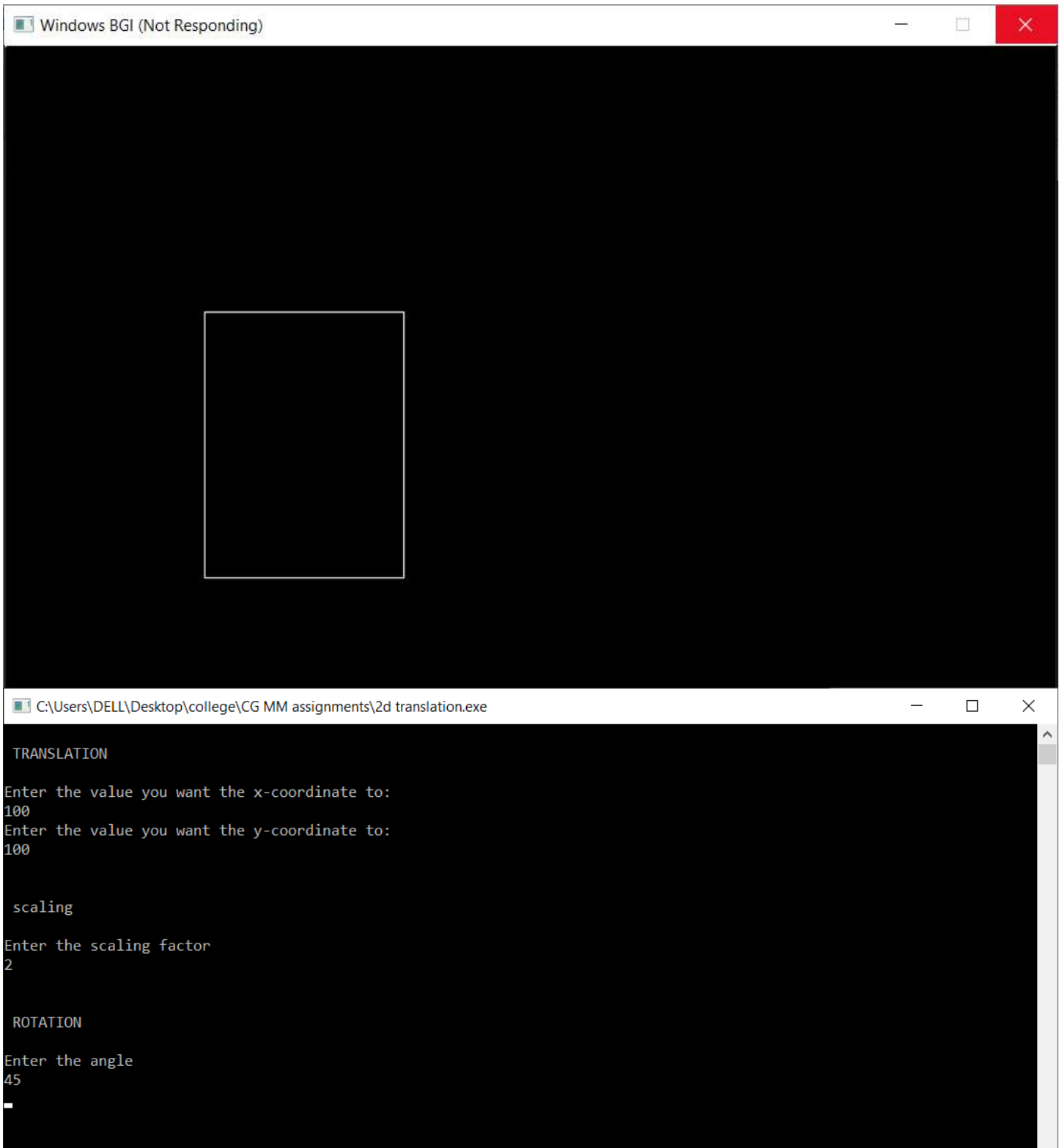
```
C:\Users\DELL\Desktop\college\CG MM assignments\2d translation.exe

TRANSLATION
Enter the value you want the x-coordinate to:
100
Enter the value you want the y-coordinate to:
100

scaling
Enter the scaling factor
2
```

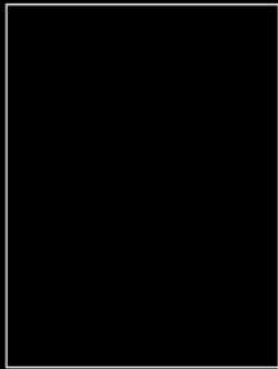


ALL TRANSFORMATIONS ON SQUARE:



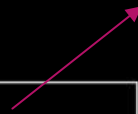


Rotation



Square

scaling



translation



VIVA-VOCE:

Q1. Given a circle C with radius 10 and center coordinates (1, 4). Apply the translation with distance 5 towards X axis and 1 towards Y axis. Obtain the new coordinates of C without changing its radius?

Ans.

Old center coordinates of C = (Xold, Yold) = (1, 4)

Translation vector = (Tx, Ty) = (5, 1)

Applying the translation equations, we have-

$$\bullet X_{\text{new}} = X_{\text{old}} + T_x = 1 + 5 = 6$$

$$\bullet Y_{\text{new}} = Y_{\text{old}} + T_y = 4 + 1 = 5$$

New center coordinates of C = (6, 5).

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix} + \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \end{bmatrix}$$

Q2. Given a line segment with starting point as (0, 0) and ending point as (4, 4). Apply 30 degree rotation anticlockwise direction on the line segment and find out the new coordinates of the line.

Ans.

Old ending coordinates of the line = (Xold, Yold) = (4, 4)

Rotation angle = $\theta = 30^\circ$

new ending coordinates of the line after rotation = (Xnew, Ynew).

Xnew

$$= X_{\text{old}} \times \cos\theta - Y_{\text{old}} \times \sin\theta$$

$$= 4 \times \cos 30^\circ - 4 \times \sin 30^\circ$$

$$= 4 \times (\sqrt{3} / 2) - 4 \times (1 / 2)$$

$$= 2\sqrt{3} - 2$$

$$= 2(\sqrt{3} - 1)$$

$$= 2(1.73 - 1)$$

$$= 1.46$$

Ynew

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \end{bmatrix}$$

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ \\ \sin 30^\circ & \cos 30^\circ \end{bmatrix} \times \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} 4 \times \cos 30^\circ - 4 \times \sin 30^\circ \\ 4 \times \sin 30^\circ + 4 \times \cos 30^\circ \end{bmatrix}$$

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} 4 \times \cos 30^\circ - 4 \times \sin 30^\circ \\ 4 \times \sin 30^\circ + 4 \times \cos 30^\circ \end{bmatrix}$$

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} 1.46 \\ 5.46 \end{bmatrix}$$

$$= X_{old} \times \sin\theta + Y_{old} \times \cos\theta$$

$$= 4 \times \sin 30^\circ + 4 \times \cos 30^\circ$$

$$= 4 \times (1/2) + 4 \times (\sqrt{3}/2)$$

$$= 2 + 2\sqrt{3}$$

$$= 2(1 + \sqrt{3})$$

$$= 2(1 + 1.73)$$

$$= 5.46$$

New ending coordinates of the line after

rotation = (1.46, 5.46).

Q3. Given a square object with coordinate points A(0, 3), B(3, 3), C(3, 0), D(0, 0). Apply the scaling parameter 2 towards X axis and 3 towards Y axis and obtain the new coordinates of the object.

Ans.

Old corner coordinates of the square = A (0, 3), B(3, 3), C(3, 0), D(0, 0)

Scaling factor along X axis = 2

Scaling factor along Y axis = 3

For Coordinates A(0, 3)

- $X_{new} = X_{old} \times S_x = 0 \times 2 = 0$

- $Y_{new} = Y_{old} \times S_y = 3 \times 3 = 9$

For Coordinates B(3, 3)

- $X_{new} = X_{old} \times S_x = 3 \times 2 = 6$

- $Y_{new} = Y_{old} \times S_y = 3 \times 3 = 9$

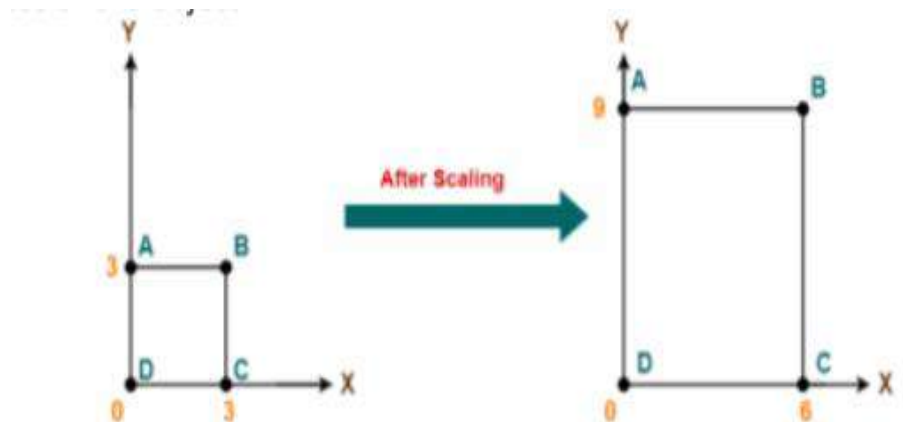
For Coordinates C(3, 0)

- $X_{new} = X_{old} \times S_x = 3 \times 2 = 6$

- $Y_{new} = Y_{old} \times S_y = 0 \times 3 = 0$

For Coordinates D(0, 0)

- $X_{new} = X_{old} \times S_x = 0 \times 2 = 0$



- $Y_{new} = Y_{old} \times S_y = 0 \times 3 = 0$

Q4. Given a triangle with coordinate points A(3, 4), B(6, 4), C(5, 6). Apply the reflection on the X axis and obtain the new coordinates of the object.

Ans.

For Coordinates A(3, 4)

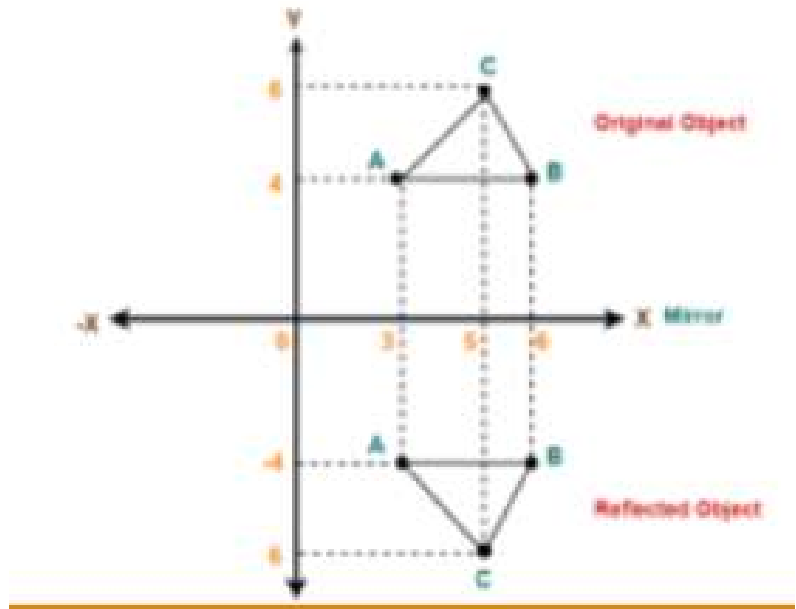
- $X_{new} = X_{old} = 3$
- $Y_{new} = -Y_{old} = -4$

For Coordinates B(6, 4)

- $X_{new} = X_{old} = 6$
- $Y_{new} = -Y_{old} = -4$

For Coordinates C(5, 6)

- $X_{new} = X_{old} = 5$
- $Y_{new} = -Y_{old} = -6$



Q5. Given a triangle with points (1, 1), (0, 0) and (1, 0). Apply shear parameter 2 on X axis and 2 on Y axis and find out the new coordinates of the object.

Ans.

- Old corner coordinates of the triangle = A (1, 1), B(0, 0), C(1, 0)
- Shearing parameter towards X direction (Sh_x) = 2
- Shearing parameter towards Y direction (Sh_y) = 2

Shearing in X AxisFor Coordinates A(1, 1)

$$X_{new} = X_{old} + Sh_x \times Y_{old}$$

$$Y_{old} = 1 + 2 \times 1 = 3$$

$$Y_{new} = Y_{old} = 1$$

For Coordinates B(0, 0)

$$X_{new} = X_{old} + Sh_x \times Y_{old} = 0 + 2 \times 0 = 0$$

$$Y_{new} = Y_{old} = 0$$

Shearing in X AxisFor Coordinates C(1, 0)

- $X_{\text{new}} = X_{\text{old}} + S_{hx} \times Y_{\text{old}} = 1 + 2 \times 0 = 1$

- $Y_{\text{new}} = Y_{\text{old}} = 0$

New coordinates of the triangle after shearing in X axis = A (3, 1), B(0, 0), C(1, 0)



EXPERIMENT 6B

COMPUTER GRAPICS AND MULTIMEDIA

Aim

Write C Programs for the implementation of 2D and 3D transformations.

Syeda Reeha Quasar
14114802719
3C7

EXPERIMENT 6B

AIM:

Write C Programs for the implementation of 2D and 3D transformations.

THEORY:

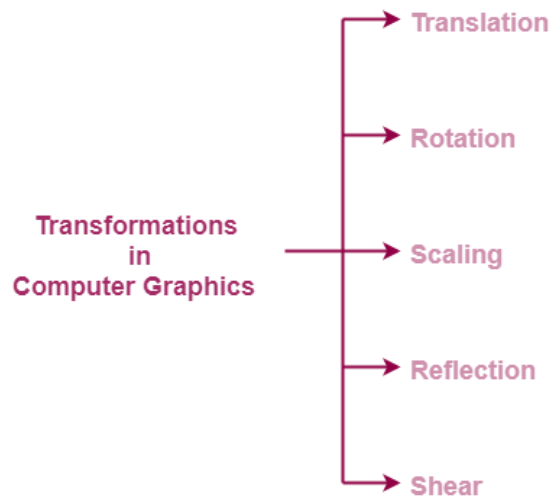
2D TRANSFORMATIONS

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.

In computer graphics, various transformation techniques are-

1. Translation
2. Rotation
3. Scaling
4. Reflection
5. Shear



In this experiment, we will discuss about 2D Reflection and Shearing in Computer Graphics.

2D Reflection in Computer Graphics-

- Reflection is a kind of rotation where the angle of rotation is 180 degree.
- The reflected object is always formed on the other side of mirror.
- The size of reflected object is same as the size of original object.

Consider a point object O has to be reflected in a 2D plane.

Let-

- Initial coordinates of the object O = (X_{old}, Y_{old})
- New coordinates of the reflected object O after reflection = (X_{new}, Y_{new})

Reflection On X-Axis:

This reflection is achieved by using the following reflection equations-

- $X_{new} = X_{old}$
- $Y_{new} = -Y_{old}$

In Matrix form, the above reflection equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Reflection Matrix
(Reflection Along X Axis)

For homogeneous coordinates, the above reflection matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \\ 1 \end{bmatrix}$$

Reflection Matrix

(Reflection Along X Axis)

(Homogeneous Coordinates Representation)

Reflection On Y-Axis:

This reflection is achieved by using the following reflection equations-

- $X_{\text{new}} = -X_{\text{old}}$
- $Y_{\text{new}} = Y_{\text{old}}$

In Matrix form, the above reflection equations may be represented as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \end{bmatrix}$$

Reflection Matrix

(Reflection Along Y Axis)

For homogeneous coordinates, the above reflection matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \\ 1 \end{bmatrix}$$

Reflection Matrix

(Reflection Along Y Axis)

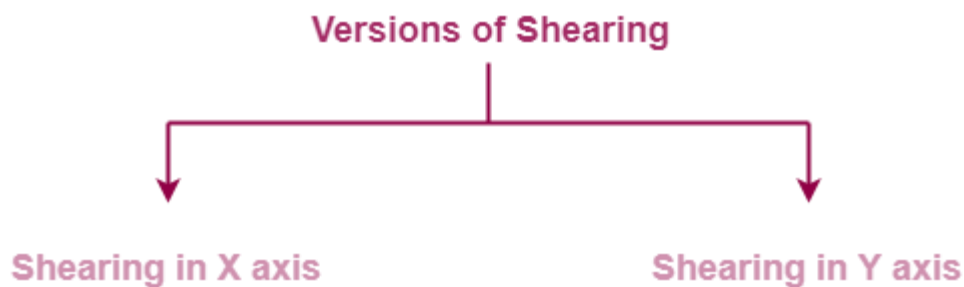
(Homogeneous Coordinates Representation)

2D Shearing in Computer Graphics-

In Computer graphics,
2D Shearing is an ideal technique to change the shape of an existing object in a two dimensional plane.

In a two dimensional plane, the object size can be changed along X direction as well as Y direction.

So, there are two versions of shearing-



1. Shearing in X direction
2. Shearing in Y direction

Consider a point object O has to be sheared in a 2D plane.

Let-

- Initial coordinates of the object O = (X_{old}, Y_{old})
- Shearing parameter towards X direction = Sh_x
- Shearing parameter towards Y direction = Sh_y
- New coordinates of the object O after shearing = (X_{new}, Y_{new})

Shearing in X Axis-

Shearing in X axis is achieved by using the following shearing equations-

- $X_{\text{new}} = X_{\text{old}} + Sh_x \times Y_{\text{old}}$
- $Y_{\text{new}} = Y_{\text{old}}$

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} 1 & Sh_x \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \end{bmatrix}$$

Shearing Matrix
(In X axis)

For homogeneous coordinates, the above shearing matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \\ 1 \end{bmatrix}$$

Shearing Matrix
(In X axis)
(Homogeneous Coordinates Representation)

Shearing in Y Axis-

Shearing in Y axis is achieved by using the following shearing equations-

- $X_{\text{new}} = X_{\text{old}}$
- $Y_{\text{new}} = Y_{\text{old}} + Sh_y \times X_{\text{old}}$

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ Sh_y & 1 \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \end{bmatrix}$$

Shearing Matrix
(In Y axis)

For homogeneous coordinates, the above shearing matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \\ 1 \end{bmatrix}$$

Shearing Matrix
(In Y axis)
(Homogeneous Coordinates Representation)

Reflection:

SOURCE CODE:

```
#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<stdlib.h>

void refx(int x1,int x2,int x3,int y1,int y2,int y3){
    line(320,0,320,430);
    line(0,240,640,240);
    x1=(320-x1)+320;
    x2=(320-x2)+320;
    x3=(320-x3)+320;
    line(x1,y1,x2,y2);
    line(x2,y2,x3,y3);
    line(x3,y3,x1,y1);
}

void refy(int x1,int x2,int x3,int y1,int y2,int y3){
    line(320,0,320,430);
    line(0,240,640,240);
    y1=(240-y1)+240;
    y2=(240-y2)+240;
    y3=(240-y3)+240;
    line(x1,y1,x2,y2);
    line(x2,y2,x3,y3);
    line(x3,y3,x1,y1);
}
```

```
int main()
{
    int x1,y1,x2,y2,x3,y3;
    initwindow(800, 800);

    //axis
    line(320,0,320,430);
    line(0,240,640,240);

    x1=150;y1=100;
    x2=220;y2=220;
    x3=220;y3=110;

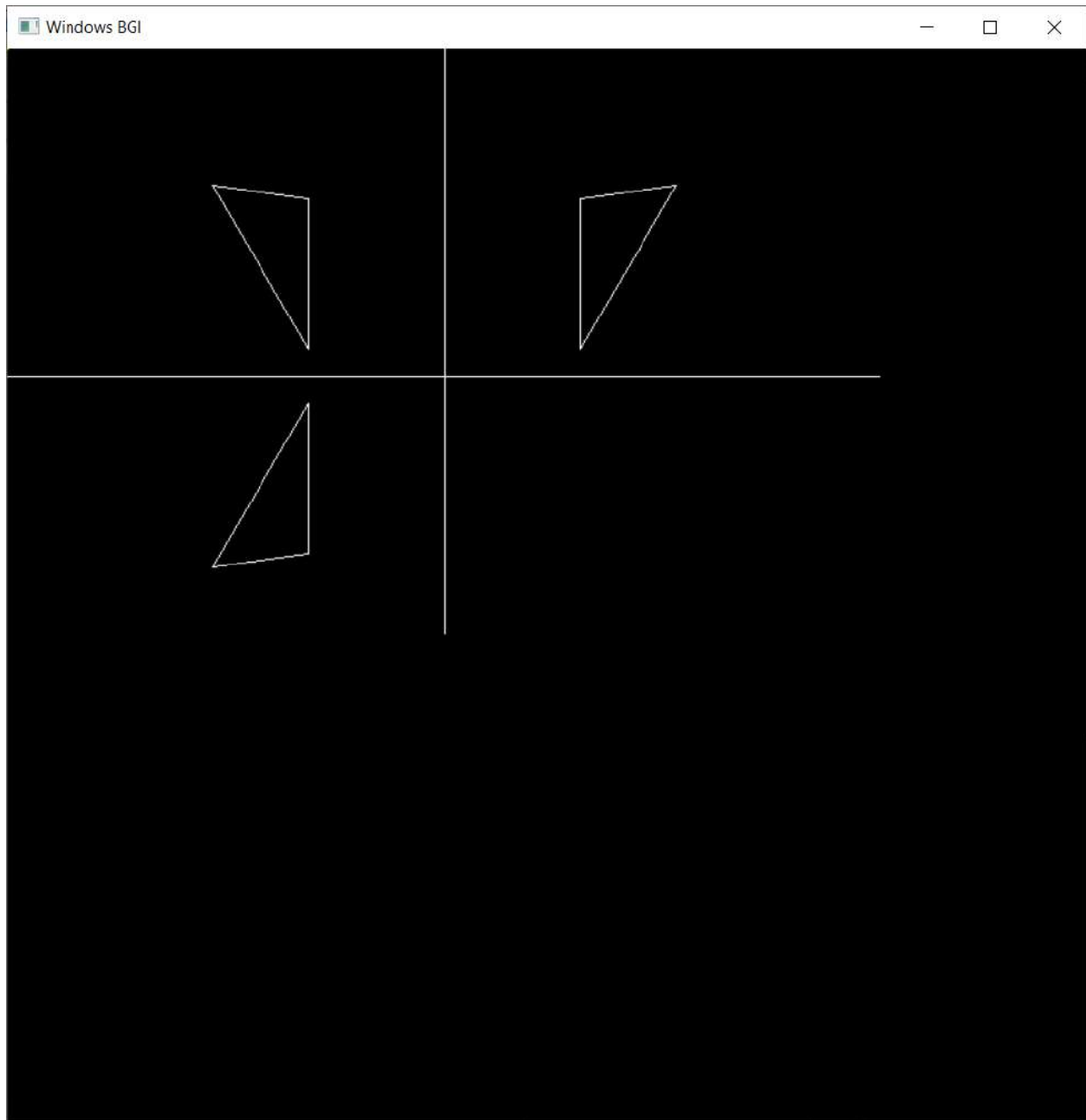
    // triangle
    line(x1,y1,x2,y2);
    line(x2,y2,x3,y3);
    line(x3,y3,x1,y1);

    refx(x1,x2,x3,y1,y2,y3);

    refy(x1,x2,x3,y1,y2,y3);
    getch();

    closegraph();
    return 0;
}
```

OUTPUT:



Shearing:

SOURCE CODE:

```
#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<stdlib.h>

void shearx(int x,int x1,int x2,int y,int y1,int y2,int shearf){
    setcolor(RED);
    x=x+ y*shearf;
    x1=x1+ y1*shearf;
    x2=x2+ y2*shearf;
    line(x,y,x1,y1);
    line(x1,y1,x2,y2);
    line(x2,y2,x,y);
}

void sheary(int x,int x1,int x2,int y,int y1,int y2,int shearf){
    setcolor(GREEN);
    y=y+ x*shearf;
    y1=y1+ x1*shearf;
    y2=y2+ x2*shearf;
    line(x,y,x1,y1);
    line(x1,y1,x2,y2);
    line(x2,y2,x,y);
}
```

```
}
```

```
int main()
```

```
{
```

```
    int x,y,x1,y1,x2,y2,shearf;
```

```
    initwindow(800,800);
```

```
    x=100;y=200;
```

```
    x1=200;y1=100;
```

```
    x2=300;y2=200;
```

```
    shearf=2;
```

```
    setcolor(BLUE);
```

```
    line(x,y,x1,y1);
```

```
    line(x1,y1,x2,y2);
```

```
    line(x2,y2,x,y);
```

```
    shearx(x,x1,x2,y,y1,y2,shearf);
```

```
    sheary(x,x1,x2,y,y1,y2,shearf);
```

```
    getch();
```

```
    closegraph();
```

```
    return 0;
```

```
}
```

OUTPUT:



THEORY:

3D TRANSFORMATIONS

The Move Tool

Moves selected object or components by dragging the transform manipulator.

The move tool is located in the toolbox. The Move tool has a manipulator that will become visible when an object is selected. The manipulator consists of arrows on the three axes which are color coded (Red = X, Green = Y, Blue = Z) and a yellow box at the center of the manipulator. To use it, first choose an object you need to move. Upon selecting it, you will see a controller appeared with four handles. They are used to move the object along each axis and one to move along the planes. Also, the colour changes to indicate that it is active when you select a handle.

The Rotate Tool

Rotates selected object or components by dragging the rotation manipulator.

The rotate tool is used to rotate objects in all the three axes. Select the rotate tool available in the toolbox and select the object you need to rotate. Now you will notice four rings colour coordinated to XYZ axes. A virtual sphere is also displayed along with the rings. You will know the selected ring by the change in colour. To perform constrained rotations, use X, Y, Z rings. In order to rotate according to the view, use the outer ring. When you start rotating the object, the application rotates it based on the object's bounding box. If you need to rotate it in fixed increments, then you can make use of the snap option. For e.g., if you set the rotation degree to 15, then you can easily rotate the objects in order of 30, 45, 60, and 90-degree positions for better symmetry. The snapping can be only used from the manipulator's axis handle.

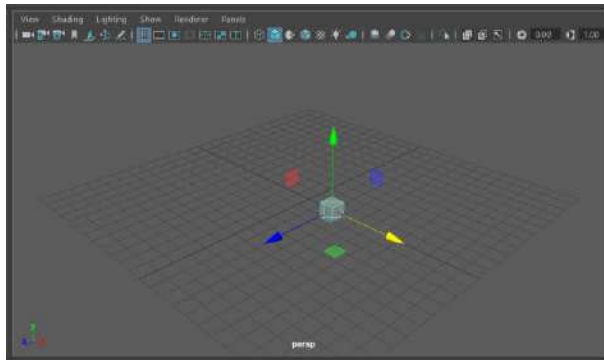
The Scale Tool

Scales selected object or components by dragging the scale manipulator.

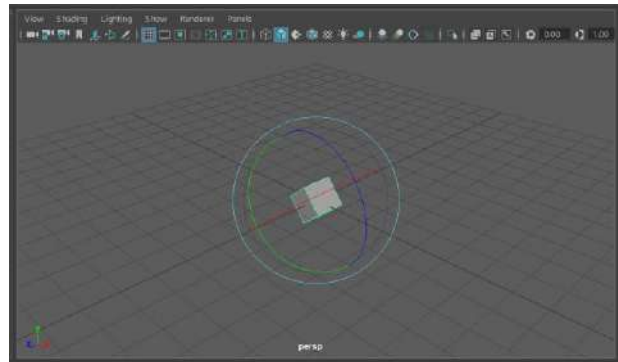
Utilize the scale tool to alter the dimension of the items by scaling uniformly in each of the three measurements. You can likewise scale unevenly in one dimension at any given moment. Snap the scale tool symbol in the tool compartment and choose the item you need to scale. Maya shows a scale controller comprising of four handles.

3D TRANSFORMATIONS:

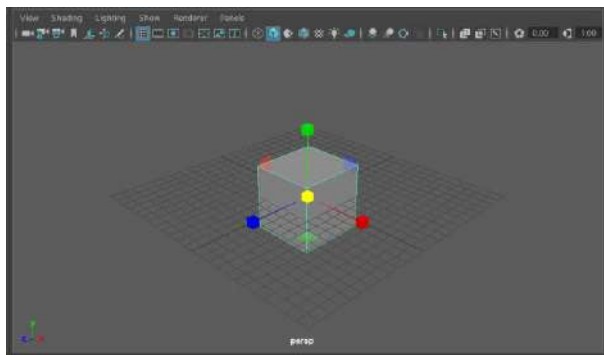
Translation



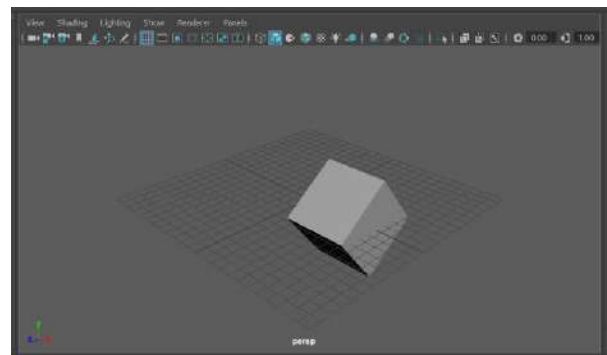
Rotation



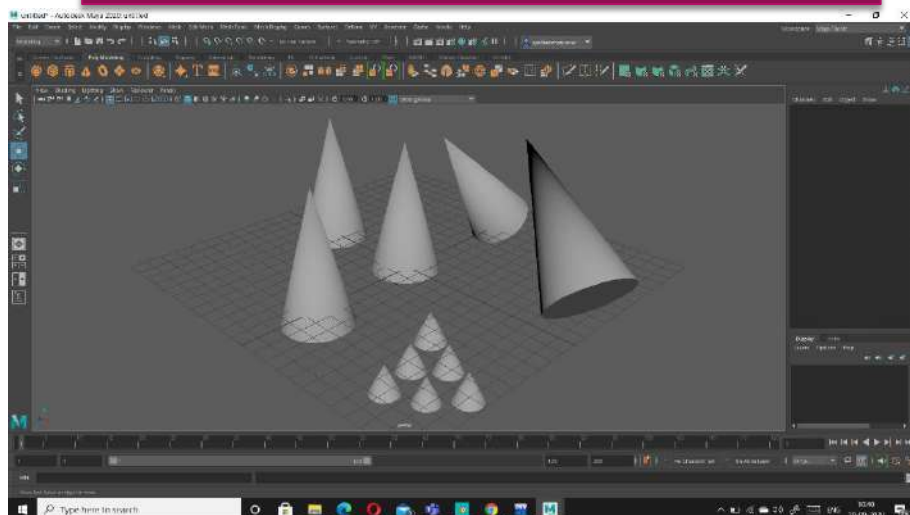
Scaling



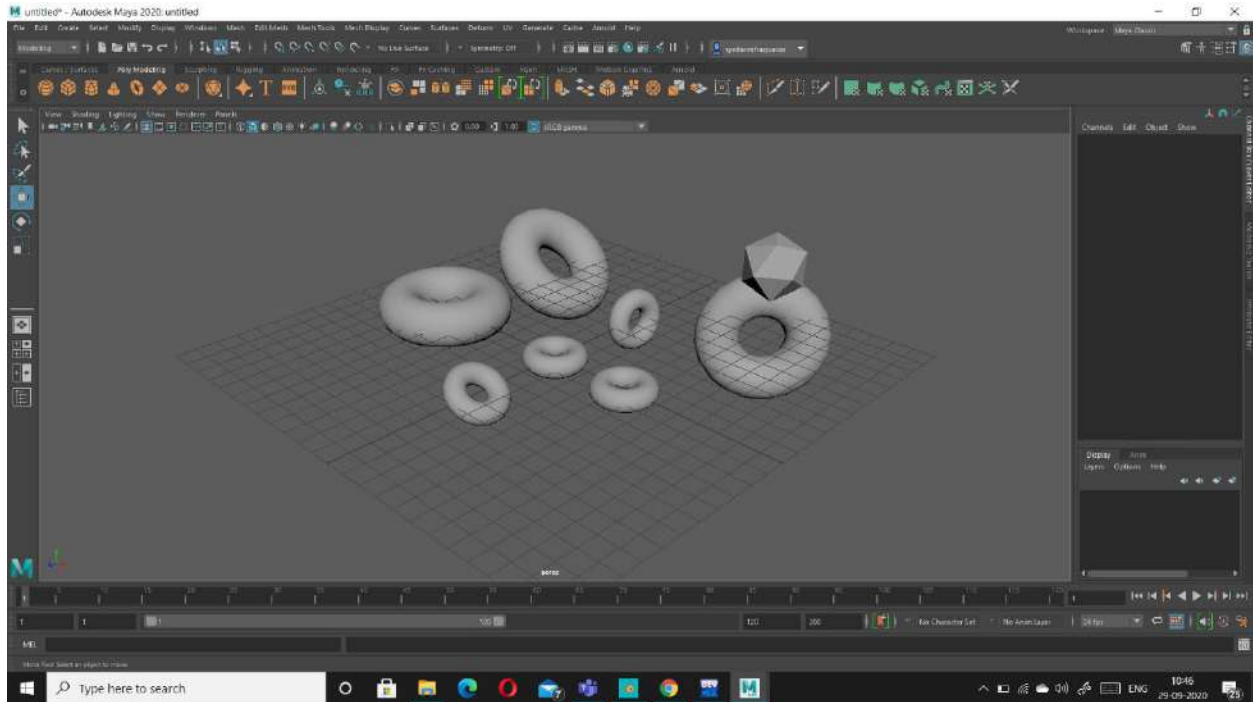
ALL 3 TRANSFORMATIONS



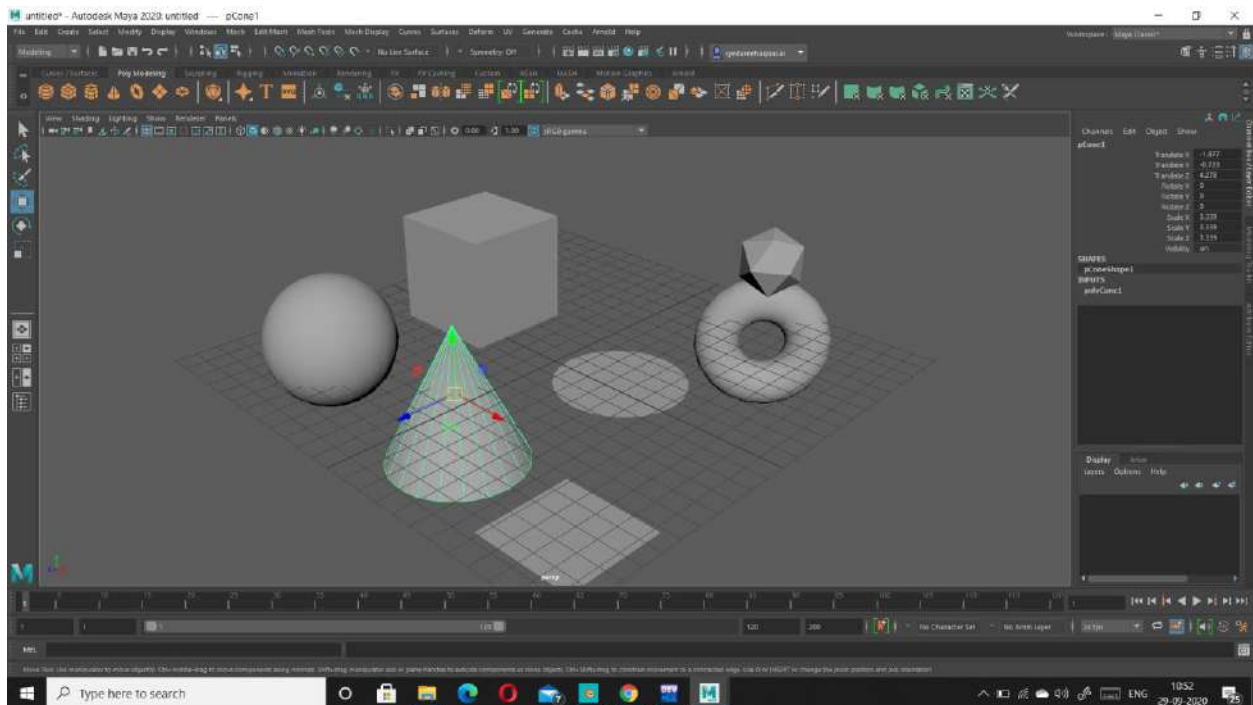
Transformations on Cone



Transformations on Polygon



Transformations on different 3D objects





EXPERIMENT 7

COMPUTER GRAPICS AND MULTIMEDIA

Aim

Write a C program to demonstrate Cohen Sutherland line clipping algorithm.

Syeda Reeha Quasar
14114802719
3C7

EXPERIMENT - 7

AIM:

Write a C program to demonstrate Cohen Sutherland line clipping algorithm.

THEORY:

The Cohen–Sutherland algorithm is a computer-graphics 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).

In the Cohen Sutherland line clipping algorithm, first of all, it is detected whether line lies inside the screen or it is outside the screen. All lines come under any one of the following categories:

1. Visible
2. Not Visible
3. Clipping Case

- 1. Visible:** If a line lies within the window, i.e., both endpoints of the line lies within the window. A line is visible and will be displayed as it is.
- 2. Not Visible:** If a line lies outside the window it will be invisible and rejected. Such lines will not display. If any one of the following inequalities is satisfied, then the line is considered invisible.

Let A (x_1, y_1) and B (x_2, y_2) are endpoints of line.

x_{min}, x_{max} are coordinates of the window.

y_{min}, y_{max} are also coordinates of the window.

$$x_1 > x_{max}$$

$$x_2 > x_{max}$$

$$y_1 > y_{max}$$

$$y_2 > y_{max}$$

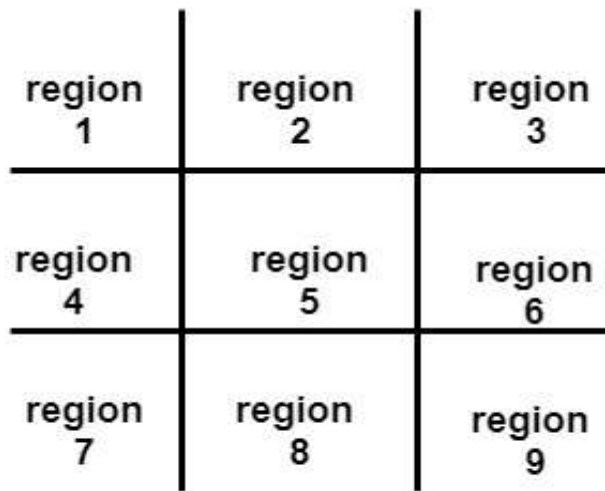
$$x_1 < x_{min}$$

$$x_2 < x_{min}$$

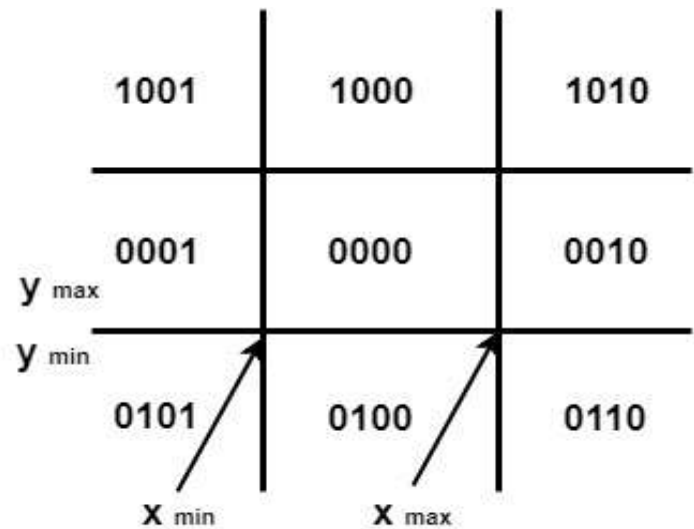
$$y_1 < y_{min}$$

$$y_2 < y_{min}$$

- 3. Clipping Case:** If the line is neither visible case nor invisible case. It is considered to be clipped case. First of all, the category of a line is found based on nine regions given below. All nine regions are assigned codes. Each code is of 4 bits. If both endpoints of the line have end bits zero, then the line is considered to be visible.



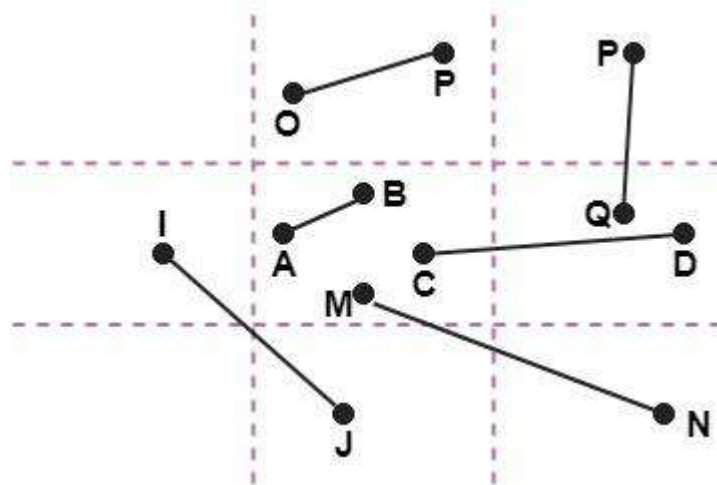
9 region



bits assigned to 9 regions

The center area is having the code, 0000, i.e., region 5 is considered a rectangle window.

Following figure show lines of various types



Line AB is the visible case

Line OP is an invisible case

Line PQ is an invisible line

Line IJ are clipping candidates

Line MN are clipping candidate

Line CD are clipping candidate

Advantage of Cohen Sutherland Line Clipping:

1. It calculates end-points very quickly and rejects and accepts lines quickly.
2. It can clip pictures much large than screen size.

Algorithm of Cohen Sutherland Line Clipping:

Step1: Calculate positions of both endpoints of the line.

Step2: Perform OR operation on both of these end-points.

Step3: If the OR operation gives 0000

Then

line is considered to be visible

else

Perform AND operation on both endpoints

If And \neq 0000

then the line is invisible

else

And=0000

Line is considered the clipped case.

Step4: If a line is clipped case, find an intersection with boundaries of the window

$$m = (y_2 - y_1) / (x_2 - x_1)$$

(a) If bit 1 is "1" line intersects with left boundary of rectangle window

$$y_3 = y_1 + m(x - X_1)$$

where $X = X_{wmin}$

where X_{wmin} is the minimum value of X co-ordinate of window

(b) If bit 2 is "1" line intersect with right boundary

$$y_3 = y_1 + m(X - X_1)$$

where $X = X_{wmax}$

where X more is maximum value of X co-ordinate of the window

(c) If bit 3 is "1" line intersects with bottom boundary

$$X_3 = X_1 + (y - y_1) / m$$

where $y = y_{wmin}$

y_{wmin} is the minimum value of Y co-ordinate of the window

(d) If bit 4 is "1" line intersects with the top boundary

$$X_3 = X_1 + (y - y_1) / m$$

where $y = y_{wmax}$

y_{wmax} is the maximum value of Y co-ordinate of the window

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()
{
    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);

    initwindow(800, 800);
    drawwindow();
    delay(500);
```

```
drawline(p1,p2);
```

```
delay(500);
```

```
cleardevice();
```

```
delay(500);
```

```
p1=setcode(p1);
```

```
p2=setcode(p2);
```

```
int 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);
```

```
getch();
```

```
closegraph();
```

```
return 0;
```

```
}
```

```
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);
```

```
}
```

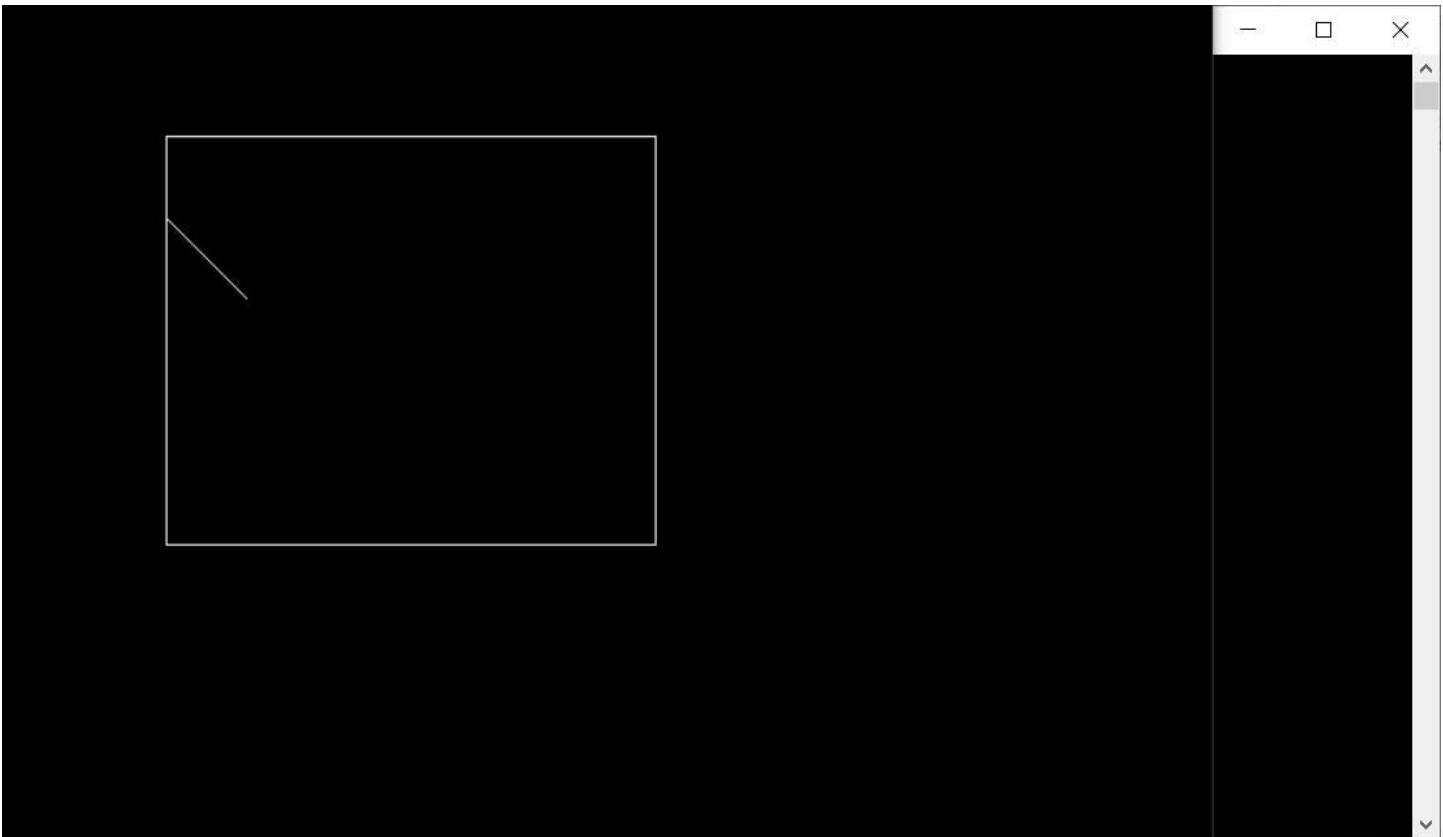
```
else
```

```
return(p1);
```

```
}
```


OUTPUT:





VIVA Voice

Q1. what is advantage of Cohen Sutherland line clipping algorithm?

Ans.

1. It calculates end-points very quickly and rejects and accepts lines quickly.
2. It can clip pictures much large than screen size.

Q2. Use the cohen sutherland line clipping algorithm to clip line P1 (70,20) and P2(100,10) against a window lower left corner(50,10) and upper right corner(80,40).

Ans.

A line with starting point P & end point Q as the mid point

$$M1 = \left(\frac{-10+50}{2}, \frac{20+10}{2} \right)$$

$$M1 = (20, 15)$$

For line PM1:

Starting point	End point	Mid point	Location
(-10,20)	(50,10)	(20,15)	Inside

(-10,20)	(20,15)	(5,17.5)	Inside
(-10,20)	(5,17)	(2.5,18.5)	Inside
(-10,20)	(2,18)	(4,19)	Inside
(-10,20)	(4,19)	(3,19.5)	Inside

For line M1Q:

Starting point	End point	Mid point	Location
(20,15)	(50,10)	(35,12.5)	Inside
(20,15)	(35,12)	(27.5,13.5)	Inside
(20,15)	(27,13)	(23.5,14)	Inside
(20,15)	(23,14)	(21.5,14.5)	Inside
(20,15)	(21,14)	(20.5,14.5)	Inside

Q3. What is the time and space complexity of Cohen Sutherland line clipping algorithm?

Ans.

- 1) Worst case time complexity: $\Theta(n)$
- 2) Average case time complexity: $\Theta(n)$
- 3) Best case time complexity: $\Theta(n)$
- 4) Space complexity: $\Theta(1)$

Q4. What do you understand by interior and exterior clipping?

Ans.

Clipping means Identifying portions of a scene that are inside (or outside) a specified region
Examples Multiple viewport on a device Deciding how much of a games world the player can see.

In other words, Clipping is the process of removing the graphics parts either inside or outside the given region.

Interior clipping removes the parts outside the given window and exterior clipping removes the parts inside the given window.

Q5. What is the difference between a window and a viewport?

Ans.

Window Port	Viewport
Window port is the coordinate area specially selected for the display.	Viewport is the display area of viewport in which the window is perfectly mapped.
Region Created according to World Coordinates.	Region Created according to Device Coordinates.
It is a region selected from the real world. It is a graphically control thing and composed of visual areas along with some of its program controlled with help of window decoration.	It is the region in computer graphics which is a polygon viewing region.
A window port can be defined with the help of a GWINDOW statement.	A viewport is defined by the GPORT command.

Q6. What are the different types of clipping algorithms?

Ans.

There are five primitive types clipping, such as point, line, polygon or are, curve and text clipping. Classical line clipping algorithms includes Cohen–Sutherland algorithm, Midpoint Subdivision algorithm, Liang Barsky and Nicholl-Lee-Nicholl algorithm.



EXPERIMENT 8

COMPUTER GRAPICS AND MULTIMEDIA

Aim

Write a C program to draw 4 point Bezier Curve.

Syeda Reeha Quasar
14114802719
3C7

EXPERIMENT - 8

AIM:

Write a C program to draw 4-point Bezier Curve.

THEORY:

Bezier curve is discovered by the French engineer **Pierre Bézier**. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as –

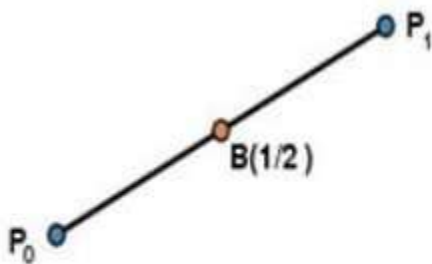
$$\sum_{k=0}^n P_k B_{ni}(t)$$

Where P_i is the set of points and $B_{ni}(t)$ represents the Bernstein polynomials which are given by –

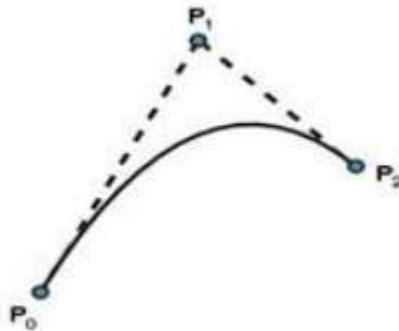
$$B_{ni}(t) = \binom{n}{i} (1-t)^{n-i} t^i$$

Where n is the polynomial degree, i is the index, and t is the variable.

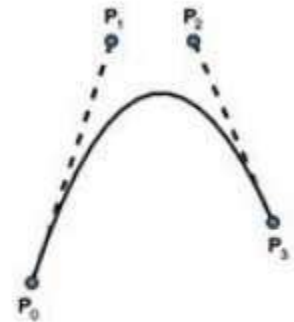
The simplest Bézier curve is the straight line from the point P_0 to P_1 . A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.



Simple Bezier Curve



Quadratic Bezier Curve



Cubic Bezier Curve

Properties of Bezier Curves

Bezier curves have the following properties –

- They generally follow the shape of the control polygon, which consists of the segments joining the control points.
- They always pass through the first and last control points.
- They are contained in the convex hull of their defining control points.
- The degree of the polynomial defining the curve segment is one less than the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
- A Bezier curve generally follows the shape of the defining polygon.

- The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.
- The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
- No straight line intersects a Bezier curve more times than it intersects its control polygon.
- They are invariant under an affine transformation.
- Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
- A given Bezier curve can be subdivided at a point $t=t_0$ into two Bezier segments which join together at the point corresponding to the parameter value $t=t_0$.

SOURCE CODE:

```
#include <stdio.h>

#include <graphics.h>

#include <math.h>

int x[4]={20,200,40,300};
int y[4]={80,30,50,400};

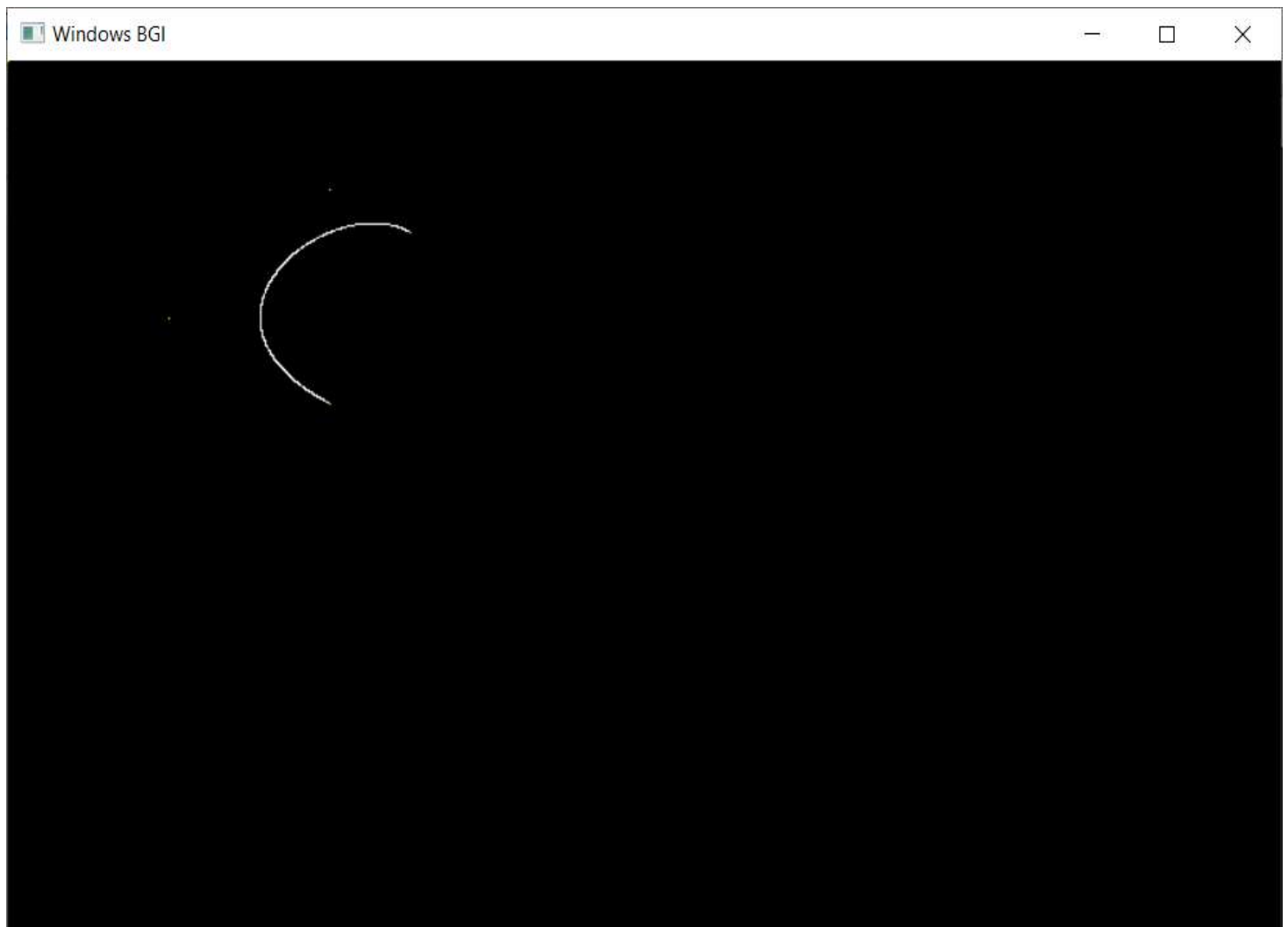
void bezier () {
    int i;
    double t,xt,yt;

    for (t = 0.0; t < 1.0; t += 0.0005) {
        xt = pow(1-t,3)x[0]+3*t*pow(1-t,2)*x[1]+3*pow(t,2)(1-t)*x[2]+pow(t,3)*x[3];
        yt = pow(1-t,3)y[0]+3*t*pow(1-t,2)*y[1]+3*pow(t,2)(1-t)*y[2]+pow(t,3)*y[3];
        putpixel (xt, yt,WHITE);
    }

    for (i=0; i<4; i++)
        putpixel (x[i], y[i], YELLOW);
}
```

```
    getch();  
}  
  
main() {  
    initwindow(1920,1080);  
    bezier ();  
    getch();  
    closegraph();  
}
```

OUTPUT:



VIVA Voice

Q1. what are control points?

Ans.

A control point is work which is aimed at checking of the compliance of the results of certain work in a business process with formulated requirements to its results. In case of non-conformity, a feed-back is arranged and the result should be corrected.

Q2. What are the applications of bezier curves?

Ans.

A Bezier curve is a parametric curve frequently used in computer graphics, animation, modeling, CAD, CAGD, and many other related fields. Bezier curves are also used in the time domain, particularly in animation and interface design, e.g., a Bezier curve can be used to specify the velocity over time of an object such as an icon moving from A to B, rather than simply moving at a fixed number of pixels per step. When animators or interface designers talk about the "physics" or "feel" of an operation, they may be referring to the particular Bezier curve used to control the velocity over time of the move in question.

Q3. what are the properties of bezier curves?

Ans.

Bezier curves have the following properties –

- They generally follow the shape of the control polygon, which consists of the segments joining the control points.
- They always pass through the first and last control points.
- They are contained in the convex hull of their defining control points.
- The degree of the polynomial defining the curve segment is one less than the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
- A Bezier curve generally follows the shape of the defining polygon.
- The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.
- The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
- No straight line intersects a Bezier curve more times than it intersects its control polygon.
- They are invariant under an affine transformation.

- Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
- A given Bezier curve can be subdivided at a point $t=t_0$ into two Bezier segments which join together at the point corresponding to the parameter value $t=t_0$.

Q4. what do you mean by approximation and interpolation points?

Ans.

Interpolation is a common way to approximate functions. Given a function with a set of points one can form a function such that for (that is that interpolates at these points). In general, an interpolant need not be a good approximation, but there are well known and often reasonable conditions where it will.

interpolation - all points of the basic figure are located on the created figure called interpolation curve segment

approximation - all points of the basic figure need not be located on the created figure called approximation curve segment.

Q5. A cubic curve is defined by points (1,1) (2,3) (4,4) (6,1). Calculate the coordinates of parametric midpoint of this curve.

Ans.

Bezier Curve :

(1,1), (2,3), (4,4), (6,1)

The equation of the Bezier Curve is given by;

$$P(u) = (1-u)^3 P_1 + 3u(1-u)^2 P_2 + 3u^2(1-u) P_3 + u^3 P_4$$

$P(0.5)$

$$= (1-0.5)^3 P_1 + 3 \times 0.5(1-0.5) P_2 + 3(0.5)^2(1-0.5) P_3 + (0.5)^3 P_4$$

$$= \frac{1}{8}(1,1) + \frac{3}{8}(2,3) + \frac{3}{8}(4,4) + \frac{1}{8}(6,1)$$

$$= \left[\frac{1}{8} \times 1 + \frac{3}{8} \times 2 + \frac{3}{8} \times 4 + \frac{1}{8} \times 6; \frac{1}{8} \times 1 + \frac{3}{8} \times 3 + \frac{3}{8} \times 4 + \frac{1}{8} \times 1 \right]$$

$$= \left[\frac{1}{8} + \frac{3}{4} + \frac{3}{2} + \frac{3}{4}; \frac{1}{8} + \frac{9}{8} + \frac{12}{8} + \frac{1}{8} \right]$$

$$= \left[\frac{1+6+6+6}{8}; \frac{1+9+12+1}{8} \right]$$

$$= \left[\frac{25}{8}; \frac{23}{8} \right]$$

$$= 2.175, 2.875$$

Q6. what is the difference between Bezier curves and B Spline curves?

Ans.

Bezier curves are parametric curves used frequently in modeling smooth surfaces in computer graphics and many other related fields. These curves can be scaled indefinitely. Linked Bezier curves contain paths that are combinations that are intuitive and can be modified. This tool is also made use of in controlling motions in animation videos. When programmers of these animations talk about the physics involved, they are in essence talking about these Bezier curves. Bezier curves were first developed by Paul de Casteljau using Casteljau's algorithm, which is considered a stable method to develop such curves. However, these curves became famous in 1962 when French designer Pierre Bezier used them to design automobiles.

B-Spline curves are considered as a generalization of Bezier curves and as such share many similarities with it. However, they have more desired properties than Bezier curves. B-Spline curves require more information such as a degree of the curve and a knot vector, and in general, involve a more complex theory than Bezier curves. They, however, possess many advantages that offset this shortcoming. Firstly, a B-Spline curve can be a Bezier curve whenever the programmer so desires. Further B-Spline curve offers more control and flexibility than a Bezier curve. It is possible to use lower degree curves and still maintain a large number of control points. B-Spline, despite being more useful are still polynomial curves and cannot represent simple curves like circles and ellipses. For these shapes, a further generalization of B-Spline curves known as NURBS is used.

- Both Bezier and B-Spline curves are used for drawing and evaluating smooth curves, especially in computer graphics and animations.
- B-Spline are considered a special case of Bezier curves
- B-Spline offer more control and flexibility than Bezier curves



EXPERIMENT 9

COMPUTER GRAPICS AND MULTIMEDIA

Aim

Using Flash/Maya perform different operations (rotation, scaling, move etc.) on objects.

Syeda Reeha Quasar
14114802719
3C7

EXPERIMENT 9

AIM:

Using Flash/Maya perform different operations (rotation, scaling move etc.) on objects.

THEORY:

Maya, is a 3D computer graphics application that runs on Windows, macOS and Linux, originally developed by Alias Systems Corporation (formerly Alia Wavefront) and currently owned and developed by Autodesk, Inc. It is used to create interactive 3D applications, including video games, animated film, TV series, or visual effects.

Moving from 2D to 3D design can be challenging since most of us design using only two dimensions. However, you 've done this before but you just don 't remembers! Maybe you haven 't realized that building objects in Maya 3D is somehow similar to building objects using Legos. The key to design in Maya 3D is to think spatially. In Maya, we design not only acknowledging width and height we also incorporate depth!

1. X, Y and Z

In Maya the X axis is the width (red), the y axis is the height (green) and the Z axis is the depth (blue). The y axis is also referred to as Y – up. The center of the coordinate system is called the origin and the coordinates are 0, 0, 0 for x, y, z.

If you want to manipulate an object in Maya you can:

a) Enter x, y and z values for translate, rotate and scale in – Channel Box

- Translate: Value – 0 in x, y and z, position the object on the center or – origin ll of the workspace.
- Rotate: If you by mistake rotate an object, set all the values to – 0. This will take the object back to its

default position.

- Scale: – 1 is the default value and it means the object 's size is 100%. If you want to make the object twice its size enter 2 in all the scale values. If you want to make an object half its size enter 0.5 in all the scale values.

b) Use the manipulators handles of the move tool, scale tool and rotate tool.

When moving & rotating objects in Maya, it 's recommended that you move or rotate the object only along one axis, this can help you maintain your objects aligned.

You can do this by pulling the arrows of the move tool or by dragging the rings

Of the rotate tool. Moving the objects from the center handle will allow you to move the object freely across the view plane and it can be hard to control its translation and rotation.

When scaling object is Maya, if you want to maintain the object 's proportions drag the center box to

scale uniformly in all directions.

2. Views

When working in a 3D environment is very important to check that the object you are building looks good in all the views. Looking to just one view can be deceiving and your object might not be positioned correctly.

In order to ease the design process Maya provides up to 4 views, one perspective view and three orthographic views.

When you look at the object from the perspective view you can rotate the camera to freely tumble around the object. However, when you use orthographic views (top, side & front) they only focus in 2 axes at the time, so you can't tumble around (unless you use the tumble tool).

3. Polygonal Modeling vs NURBS Modeling Polygons:

Polygons are the most basic geometry type in Maya they are straight sided shapes of 3 sides or more. Polygons have faces, edges & vertices, some examples of polygons or primitives' types are spheres, cubes, cylinders, cones, and planes.

Because polygon 's surfaces can be directly extruded, scaled and positioned designers prefer them to create characters in Maya.

NURBS Curves and NURBS Surfaces:

NURBS curves stand for Non-Uniform Rational B-Splines. NURBS curves are very handy since you can draw them in Maya or import them from a vector program like Adobe Illustrator.

NURBS primitives or surfaces, use UV coordinate space and you can modify them by trimming away portions of their forms, beveling their edges, or by sculpting the mesh to different shapes using the Maya artisan sculpting tools

NURBS are preferred for constructing organic 3D forms because of their smooth & natural characteristics.

Pivot

All transformations to an object are relative to the pivot point. The object 's pivot affects scale since the object will scale out from origin toward the pivot point. The pivot also affects the object 's rotation/animation because an object rotates around the pivot point.

4. Subdivisions Polygon Subdivisions:

If a polygon has less subdivisions, for example a value of 5, it will have less detail and a rougher surface.

If a polygon has more subdivisions, for example a value of 20, it will have more detail & a smoother surface.

NURBS Subdivisions

If a NURBS has less sections & spans, for example a value of 2, it will have less detail and a rougher surface

If a NURB has more sections & spans, for example a value of — 20, it will have more detail and a smoother

5. UV Coordinate Space:

Surfaces in Maya have their own coordinate space. UV Coordinate Space is useful when working with curve-on-surfaces and positioning textures.

6. Nodes

Every element in Maya is built with a single or a series of nodes. Nodes define all attributes like lighting, shading and geometry.

For example a primitive polygon, such as a sphere, is built from several nodes: a creation node that records the options that created the sphere, a transform node that records how the object is moved, rotated, and scaled, and a shape node that stores the positions of the spheres control points.

Shape nodes:

Holds an object's geometry attributes or attributes other than the object's transform node attributes.

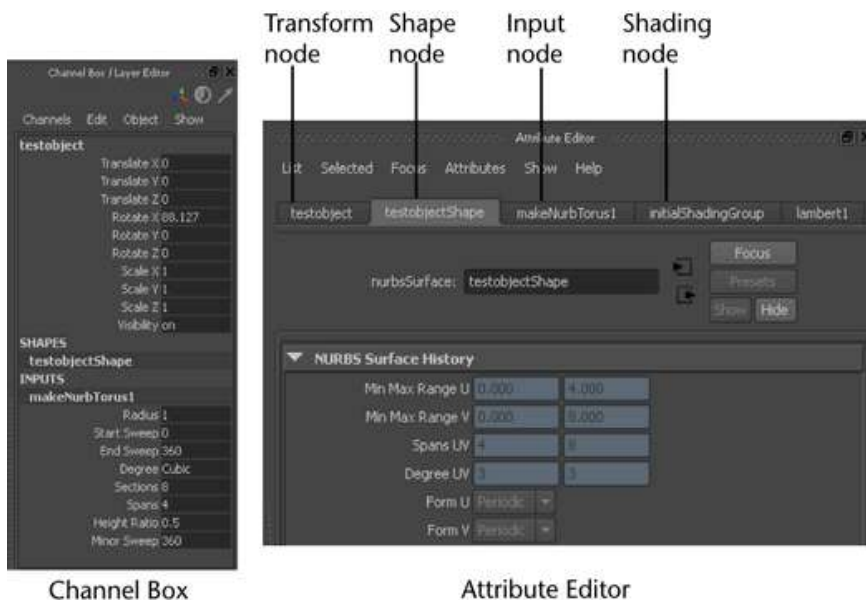
Rendering nodes:

Materials and textures each have nodes containing attributes that control their look. Texture placement nodes have attributes that control how a texture is fitted onto a surface.

Attributes

An attribute is a position associated with a node that can hold a value or a connection to another node. Attributes control how a node works. For example, a transform node has attributes for the amount of rotation in X, Y, and Z. You can set attributes to control practically every aspect of your animation.

There are many ways to set attributes in Maya: with the Attribute Editor, the Channel Box, the Attribute Spread Sheet, menu selections, and MEL.



Every node is created with certain default attributes. Some attributes (such as Opacity and Color of particle objects) are added dynamically when you need them.

You can also add your own attributes to any node to store information. This is often useful for animation expressions and scripts, and can be used to control several normal attributes using one custom attribute.

Attention: Node and attribute names in Maya cannot begin with a number.

Keyable attributes

Animation in Maya is not limited to making things move. You can animate practically any attribute of any node in Maya. Attributes that control how a surface is constructed, or the look of a texture, or the influence of a deformer or physical force, can all change over time.

Construction history

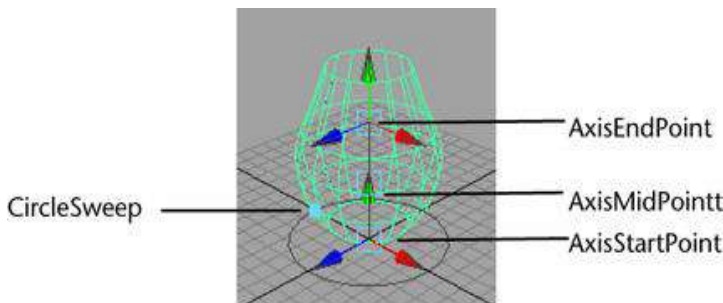
As you work in Maya, most of your actions create nodes in the Construction history of the objects you work on. At each point in your work, the current scene is the result of all the nodes you've created so far.

Manipulating nodes and attributes

Much of working in Maya involves directly manipulating nodes and attributes using manipulators.

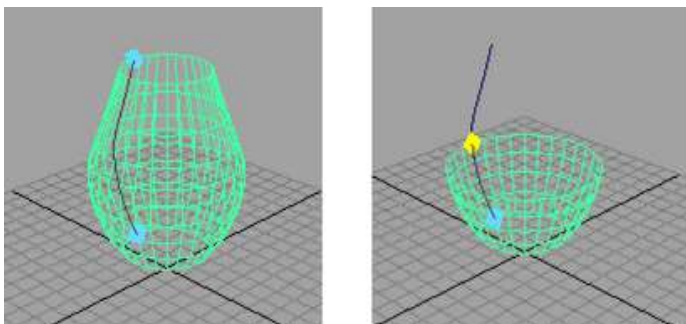
Manipulators are visual objects that let you accomplish complex tasks easily, concretely, and visually by dragging handles and seeing the results immediately.

Using the revolve example above, you can select the revolve node and edit its attributes (how it creates the surface) visually by showing its manipulator with the Show Manipulator Tool.



This lets you control attributes (such as how far around the centerpoint the surface goes) simply by dragging a handle.

You can also show manipulators for individual attributes to edit their values visually.



- 9. As a Scene hierarchy: A hierarchical list of nodes.

This shows which nodes are parents and children of other nodes. You can use the Outliner to view and edit the scene in this type of view.

The Outliner shows a hierarchical list of all objects in the scene.

- As a Dependency graph: A graph of connections between nodes.

This shows which nodes provide input or output to other nodes. You can use the Hypergraph or Node Editor to view and edit the scene in this type of view.

The Hypergraph shows a network of boxes representing nodes, and the lines that connect them to represent their relationships.

10. Parenting

The child objects have independent nodes, for example a child can rotate by itself without rotating the parent. However, if the parent is rotated or moved the child follows.

Object Manipulation Tools

Obviously, these are all operations that sound relatively self-explanatory, but let's take a look at some of the technical considerations.

There are two different ways to bring up the translate, scale, and rotate tools:

- First, they can be accessed from the toolbox panel (pictured above) on the left side of your viewport.
- The second (preferred method) is to use keyboard hotkeys. During the modeling process, you'll be switching between tools constantly, so it's a good idea to learn the commands as quickly as possible.

With an object selected, use the following hotkeys to access Maya's translate, rotate, and scale tools:

Translate - w

Rotate - e

Scale - r

To exit any tool, hit q to return to selection mode.

Translate (Move)

Select the object you created and strike the w key to bring up the translation tool.

When you access the tool, a control handle will appear at your object's central pivot point, with three arrows aimed along the X, Y, and Z axes.

To move your object away from the origin, click any one of the arrows and drag the object along that axis. Clicking anywhere on the arrow or shaft will constrain movement to the axis it represents, so if you only want to move your object vertically, simply click anywhere on the vertical arrow and your object will be constrained to vertical movement.

If you'd like to translate the object without constraining motion to a single axis, clicking in the yellow square at the center of the tool to allow free translation. When moving an object on multiple axes, it's

often beneficial to switch into one of your orthographic cameras (by clicking spacebar, in case you'd forgotten) for more control.

Scale

The scale tool functions almost exactly like the translate tool.

To scale along any axis, simply click and drag the (red, blue, or green) box that corresponds to the axis you'd like to manipulate.

To scale the object globally (simultaneously on all axes), click and drag the box located at the center of the tool. Simple as that!

Rotate

As you can see, the rotation tool appears and operates slightly different from the translate and scale tools.

Like translate and scale, you can constrain rotation to a single axis by clicking and dragging any of the three inner rings (red, green, blue) visible on the tool.

You can freely rotate the object along multiple axes, by simply clicking and dragging in the gaps between rings, however, you're afforded a lot more control by rotating an object one axis at a time.

Finally, by clicking and dragging on the outer ring (yellow), you can rotate an object perpendicular to the camera.

With rotation, there are times when a bit more control is necessary — on the next page we'll look at how we can use the channel box for precise object manipulation.

05

of 05

Using the Channel Box for Precision

In addition to the manipulator tools we've just introduced, you can also translate, scale, and rotate your models using precise numeric values in the channel box.

The channel box is located in the upper right section of the interface and functions exactly like the Inputs tab that we introduced in lesson 1.3.

There are quite a few instances where numeric values can be useful:

- Scale in Maya is based on real-world units (centimeters by default), and many of Maya's lights behave more realistically when objects are modeled using approximate real-world scale. This means if you're modeling a table that's supposed to be four feet tall, it should be scaled to approximately 162 cm.
- The channel box can also be useful if you need to space objects evenly, set fractional scales (double, half, etc.), align objects along an axis, or set exact angles for rotation (45 degrees, 90, 180, 360, etc.).

Like in the inputs tab, values can be keyed manually or by using the click + middle mouse drag gesture we introduced previously.

Finally, the channel box can be used to rename any object in your scene, including models, cameras, lights, or curves. It's a very good idea to get in the practice of naming your objects for better organization.

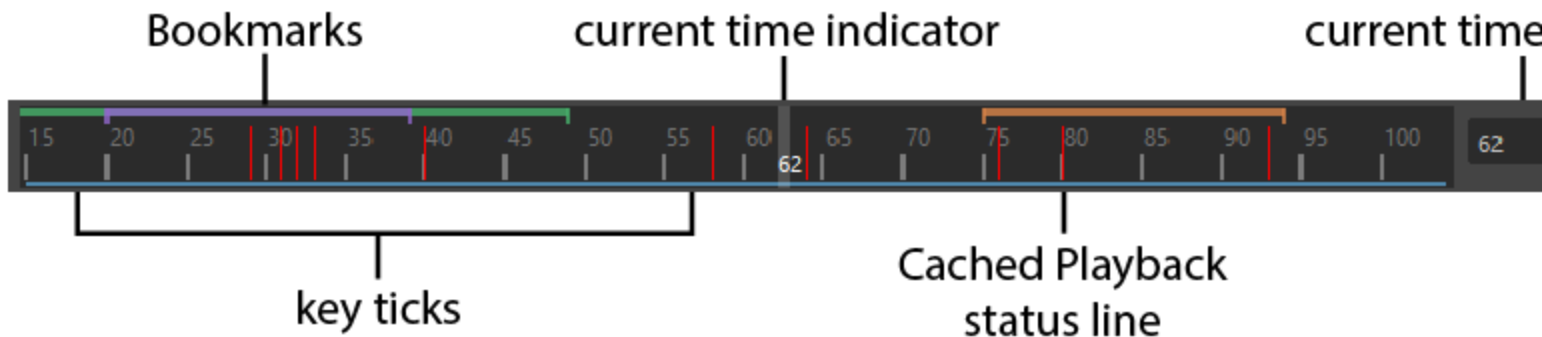
ANIMATION


- *Keyframe animation* lets you transform objects or skeletons over time by setting keyframes. For example, you can keyframe the joints and IK handles of a character's arm to create an animation of its arm waving. For more information, see [Keyframe Animation](#).
- *Driven key animation* lets you link and drive the attributes of one object with those of another object by setting driven keys. For example, you can key a character's X and Z translations as Driver attributes and a door model's Y rotation as the Driven attribute to create an animation of a character and a swinging door. For more information, see [Driven keys](#).
- *Nonlinear animation* lets you split, duplicate, and blend animation clips to achieve the motion effects that you want. For example, you can use nonlinear animation to create a looping walk cycle for one of your characters. For more information, see [Nonlinear Animation](#).
- *Path animation* lets you set a curve as an animation path for an object. When you attach an object to a motion path, it follows the curve during its animation. For example, when you assign a car model to a motion path that follows a road in your scene, the car follows the road when you play the animation. For more information, see [Path Animation](#).
- *Motion capture animation* lets you use imported motion capture data to apply realistic motion to the characters in your scene. For example, you can use the captured motion of a horse to animate the skeleton of a quadruped model. For more information, see [Motion Capture Animation](#).
- *Layered animation* lets you create and blend animation on separate layers. You can modify an animation sequence on layers without permanently altering the original, or simply organize your keyframe animation onto layers. See [Animation Layers](#).
- *Dynamic animation* lets you create realistic motion using the rules of physics to simulate natural forces. For example, you can use Maya® Dynamics™ to create effects such as sparks spraying from a welding torch or hail falling from the sky.
- *Expressions* are instructions that you can type to animate attributes. For example, you can write an expression formula that animates the flapping of a bird's wings. For more information, see [Animation expressions](#).

Animation controls

Between the Range slider and the Animation Preferences button are the Current character control features and the Auto Key button.

The Time Slider controls the playback range, keys, and breakdowns within the playback range.



Tip: Resize the Time Slider to make it taller or more compact by hovering over the top edge of the Time Slider until the splitter cursor  appears and dragging it up or down. There is an Option Variable to configure the Time Slider splitter bar sensitivity if it is distracting while working in the Viewport: `optionVar -iv "splitterHandleMargin" 3.`

Current Time Indicator

The Current Time Indicator is a gray block on the Time Slider. You can drag it to move forward and backward in your animation.


By default, dragging in the Time Slider updates only the active view. All views can be set to update by changing the Playback settings to Update View All in the Time Slider preferences (Window > Settings/Preferences > Preferences).

Note: See Customize the Time Slider if you want to change the colors and opacity of elements in the Time Slider.

Right-click the Time Slider to access the Animation controls menu, which provides easy access to common operations.

Time Slider Bookmarks

Time Slider Bookmarks are colored tags along the top of the Time Slider that flag events in your scene. Bookmarks are a way you can draw attention to specific moments in time which is useful when you want to focus or highlight specific areas or moments in your scene.

Add a Bookmark to a selected time range by clicking the Bookmark icon  in the Range Slider or pressing Alt (Option) + T.

See Create a Bookmark for more.

Cached Playback status line

The Cached Playback status line is a blue stripe that displays the condition of the Cached Playback data for the current scene. Cached Playback is a method Maya uses to increase animation playback speed by storing the animation in memory and recomputing only the part of the animation that was changed, rather than updating the whole scene for the entire time range.

Whenever you edit a scene that has been cached, the modified area of the status line temporarily turns dark blue to show the segment of the animation that is out of date before becoming blue again to show that the values have been updated.

When Cached Playback encounters an unsupported node it will enter Safe Mode. While Cached Playback is in Safe Mode, the Cached Playback status line turns yellow and a warning symbol appears on the Cached Playback icon. See the Script Editor for an explanation of what is currently causing Cached Playback to enter Safe Mode. (You can find a list of currently unsupported nodes in Cached Playback unsupported nodes.)

Right-click the Cached Playback status line to turn Cached Playback off and on, flush the cache, or select a new Caching Mode. You can change the color of the Cached Playback status line in the Color Settings and set its width and placement in the Cached Playback Preferences.

See [Use Cached Playback](#) to increase playback speed for more information about Cached Playback.

Key ticks

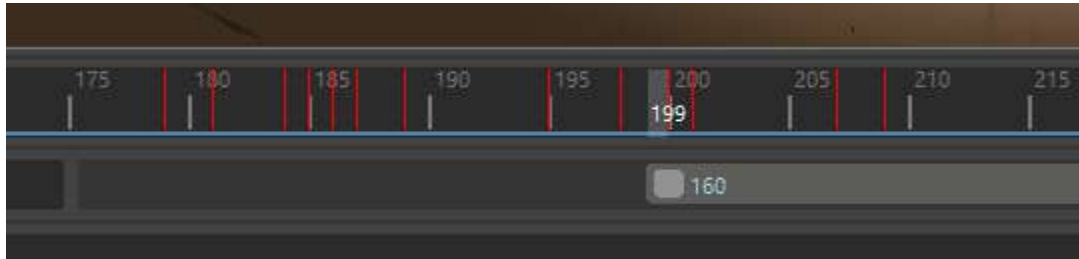
Key Ticks are red (by default) marks in the Time Slider that represent the keys you set for the selected object. Breakdowns are a special type of key displayed as green marks in the Time Slider. See [Breakdowns](#).

The visibility of Key Ticks can be turned off or on in the Preferences window. You can also set the size and color of the key ticks displayed in the Time Slider. See [Customize the Time Slider](#).

Selecting keys on the Time Slider

To select a key on the Time Slider, Shift-select it. If you need to select a *range* of keys on the Time Slider, such as when you work with Time Slider Bookmarks, Shift + drag across the Time Slider.

Turn on Auto Snap Keys in the Time Slider Preferences to ensure that your key selection snaps to the nearest whole frame.



Time units

The ruler markings and associated numbers on the Time Slider display time. To define the playback rate, select the desired Time unit in the Settings preferences (Window > Settings/Preferences > Preferences). Maya defaults to measuring time as 24 frames per second, the standard frame rate for film.

Note: By default, Maya plays your animation in seconds. You can change the Time unit type without affecting your animation's key-based behavior, but it's a good idea to specify the Time unit before you begin animating. However, expressions that use the frame variable might not work correctly if you change the Time setting.

Current time field

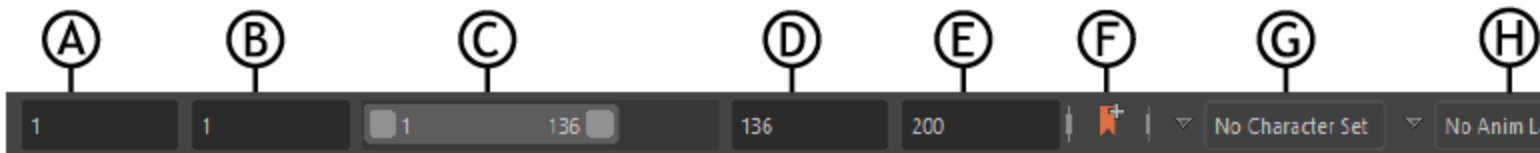
The entry field to the right of the Time Slider indicates the current time expressed in the current Time unit. You can change the current time by entering a new value. Your scene moves to that location in time, and the Current Time Indicator updates accordingly.

Audio

Right-click the Time Slider to open the Animation controls menu, where you can select the Audio rollout that gives you access to the Audio menu, where you can manage your Maya's audio settings.

When you import an audio file, its waveform shows on the Time Slider. See [Display audio on the Time Slider](#).

The Range Slider, located below the Time Slider, controls the playback range reflected in the Time Slider.



A. Animation Start Time B. Playback Start Time C. Range Slider Bar D. Playback End Time E. Animation End Time F. Time Slider Bookmark icon G. Character Set menu H. Animation Layer menu

See Playback options for information about the Framerate, Loop, Auto Key, and Animation icons that appear beside the Range Slider.

Note: Time Slider Bookmarks (F.) and Character Set menu (G.) are not available in Maya LT.

Animation Start Time

(A. in the illustration above.)

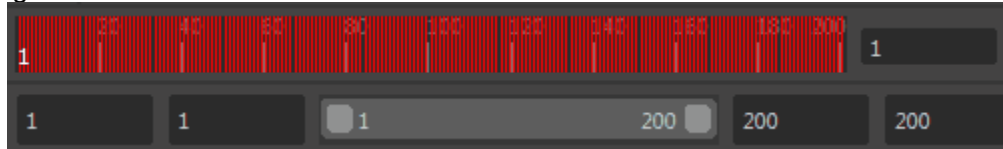
This field sets the start time of the animation.

Playback Start Time

(B. in the illustration above.)

This field shows the current start time for the playback range. You can change it by entering a new start time, including a negative value. If you enter a value that is greater than the Playback End Time, the Playback End Time is adjusted to one time unit greater than the Playback Start Time.

Range Slider bar



(C. in the illustration above.)

Drag either end of the Range Slider to manually extend or shorten the Playback Range.

Playback End Time

(D. in the illustration above.)

This field shows the current end time for the playback range. You can change it by entering a new end time. If you enter a value less than the Playback Start Time's value, the Playback Start Time is shifted to one time unit less than the Playback End Time.

You can also edit the preceding settings from the Animation Preferences window.

Note: Changes to the Range slider are undoable. You can change this setting in the Undo preferences with the Consolidate Time Changes option.

Animation End Time

(E. in the illustration above.)

This field sets the end time of the animation.

Time Slider Bookmark icon

(F. in the illustration above.)

Click  to set a Time Slider Bookmark for the selected time. See Time Slider Bookmarks for more.

Character Set menu

(G. in the illustration above.)

The Character menu is a quick way to switch the current Character Set.

Animation Layer menu

(H. in the illustration above.)

Note:

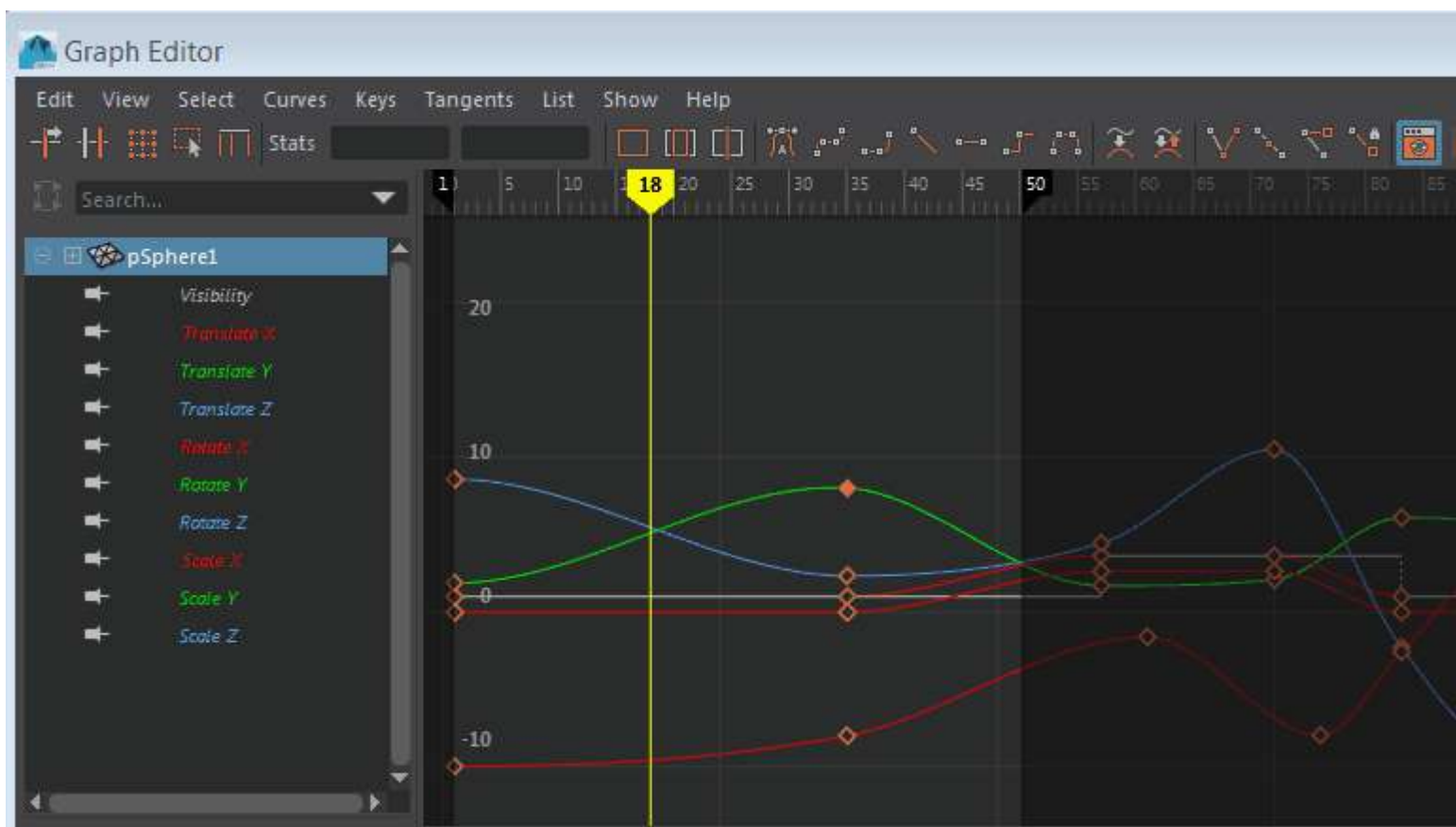
The Animation menu is a quick way to switch the current Animation Layer.

A graphical representation of interpolation between animation keys that you can modify.

To open the Graph Editor:

- From the main menu bar, select Windows > Animation Editors > Graph Editor.
- From the view menu bar, select Panels > Panel> Graph Editor.

- You can switch to the Classic Graph Editor in the Preferences window. See Animation (Settings) Preferences > Graph Editor > Graph Editor and turn on the Classic option.
- This version of Graph editor was introduced in Maya 2017, if you are running an earlier version of Maya, only the Classic Graph Editor is available.



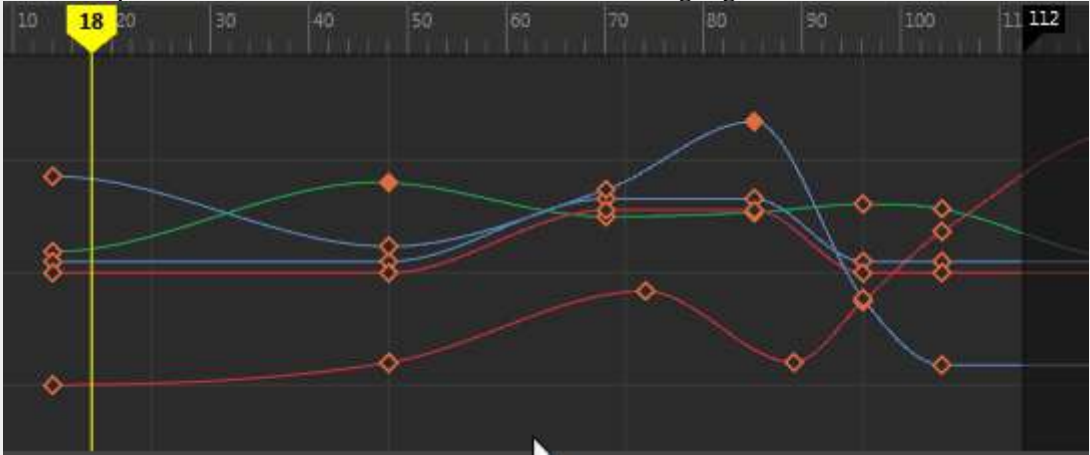
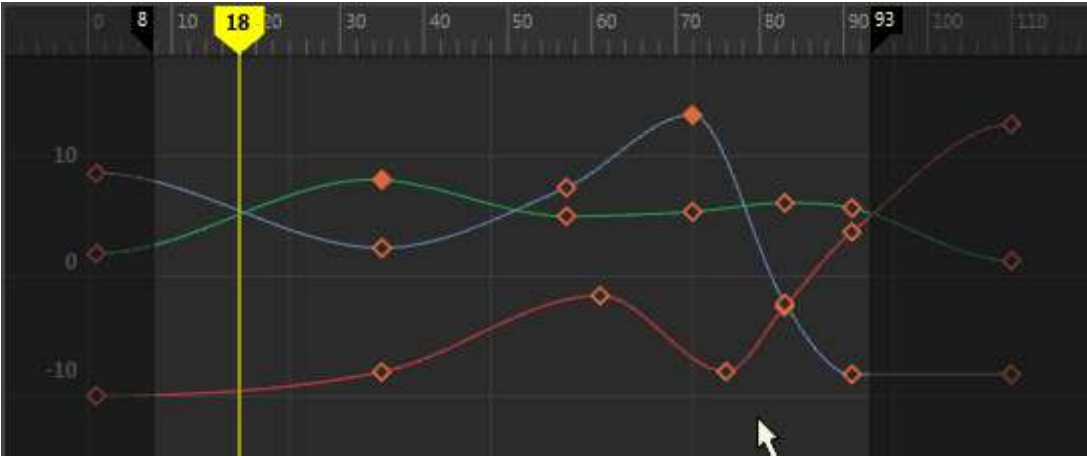
Maya's Graph Editor is an updated version of the legacy Maya Graph Editor, giving you a more intuitive approach to manipulating animation curves and keys in your scene.

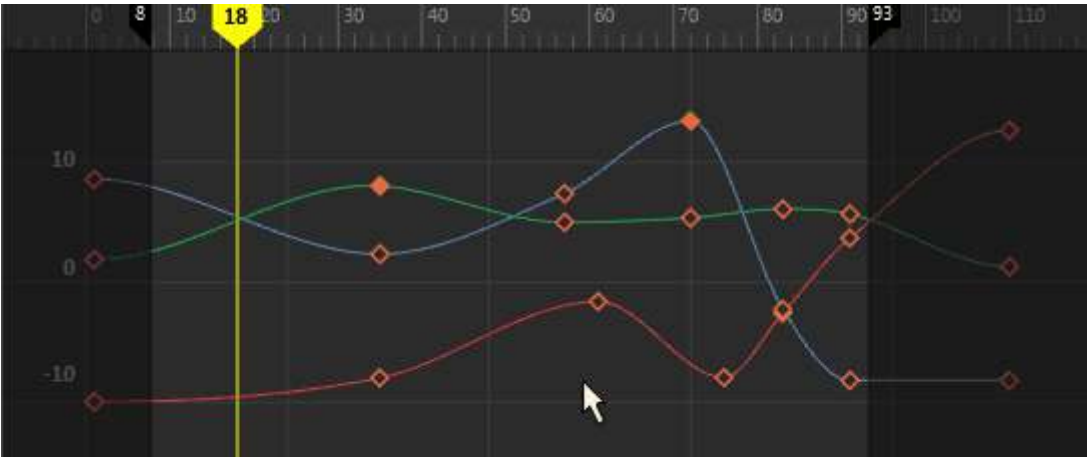
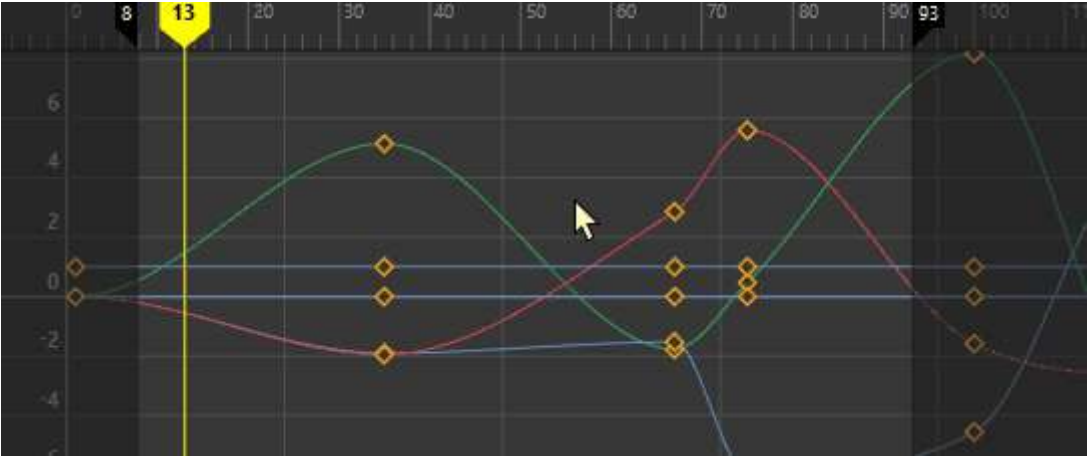
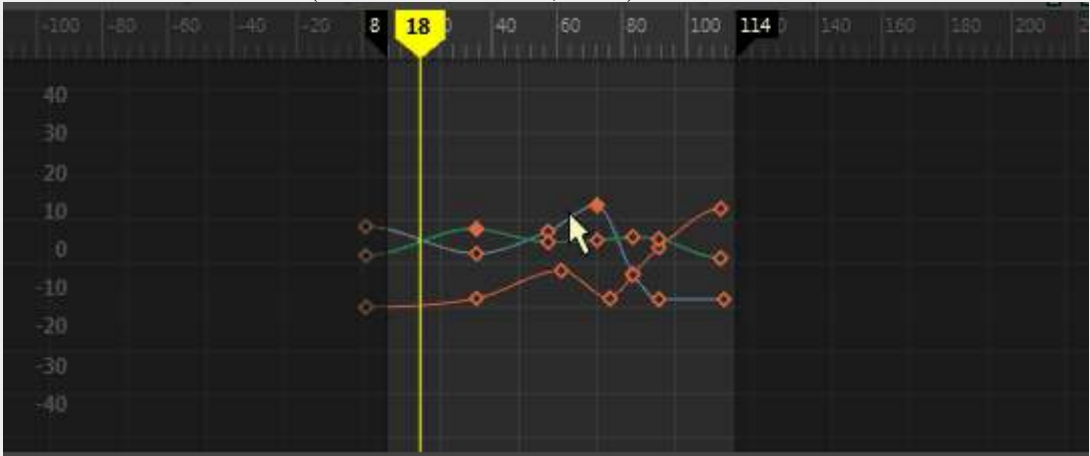
The Graph Editor is presented as graph view of scene animation so you can create, view, and modify animation curves various ways. For example, you can control interpolation between keyframes, extrapolation of curves, and change animation curves value and timing by altering the shape of animation curves using tangent handles. Note: While the Graph Editor's color scheme is designed for high visibility, you can customize the colors in the Color Settings.

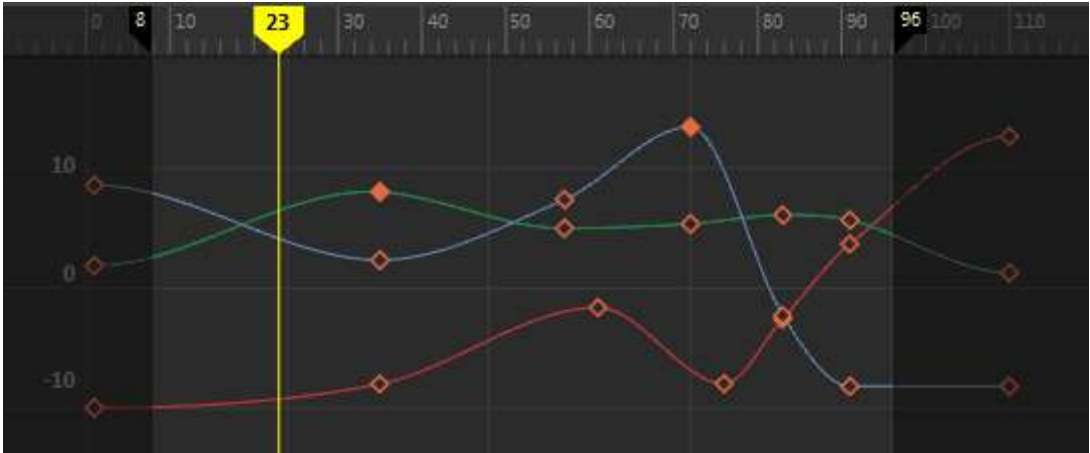
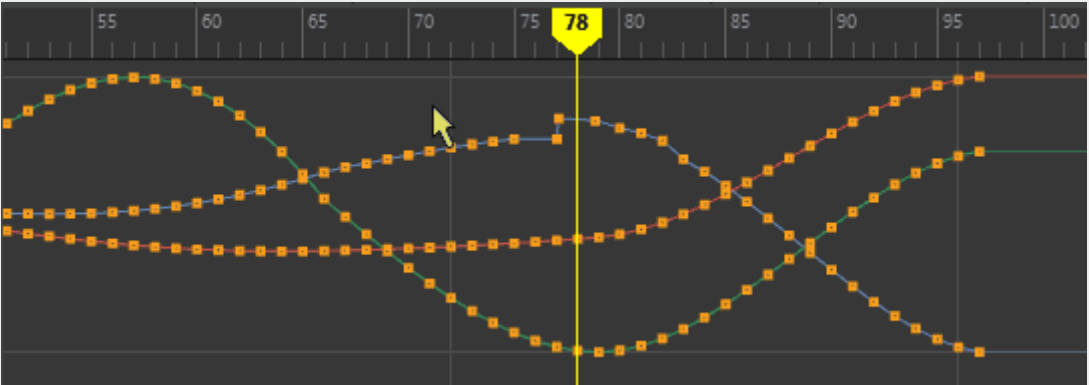
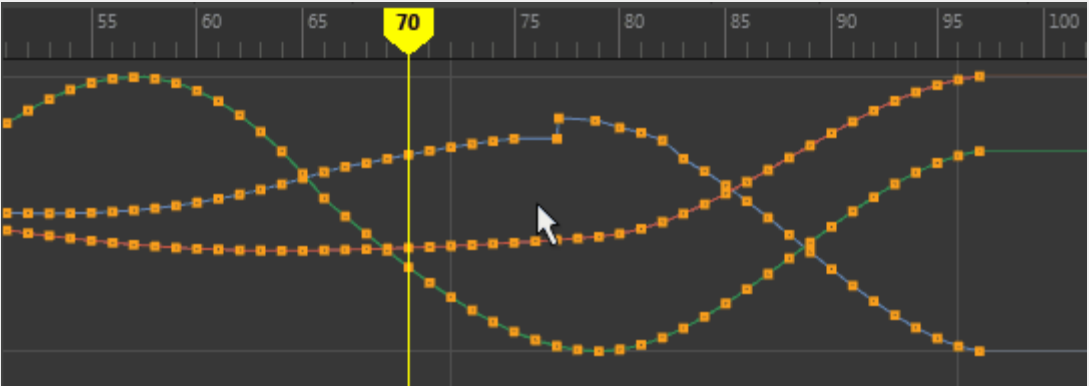
Other improvements include:

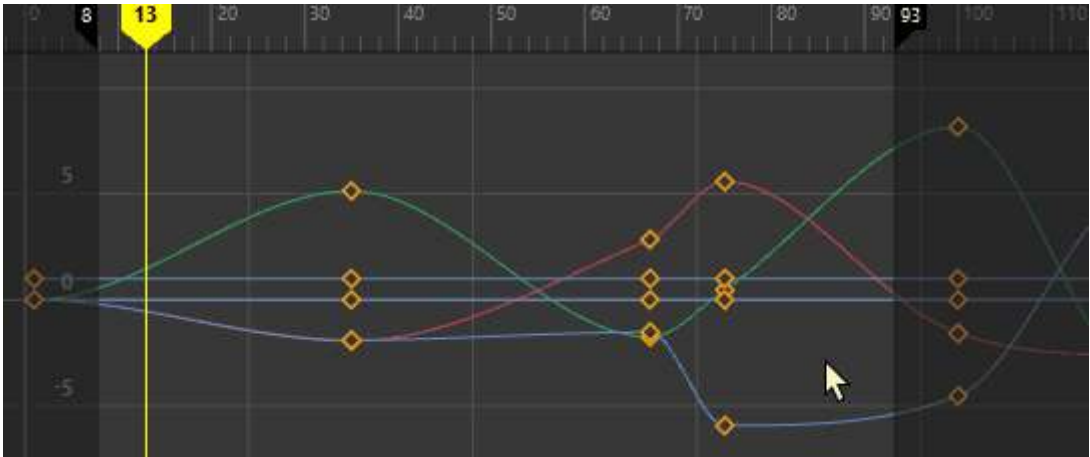
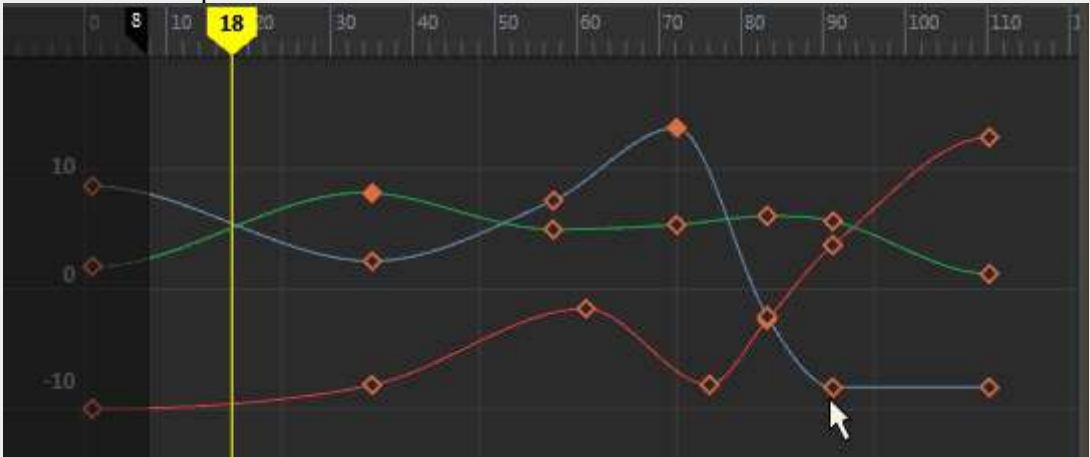


- The Move Tool is no longer needed to move keys
- Tangents are non-weighted by default
- Interactive keyframes and the ability to isolate curves
- Scrubbable Time Marker that moves anywhere you click on the Time ruler
- Resizeable play range
- Simplified view names. (Stacked Curves is now Stacked View, Display Normalized is now Normalized View and now there is an Absolute View.)
- Customizable key and curve colors (See the Graph Editor section in the Color Settings)

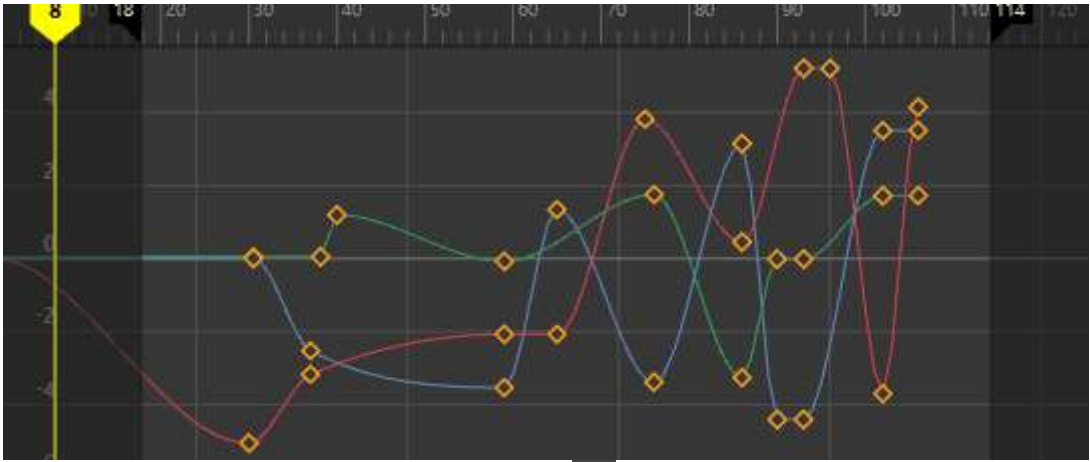

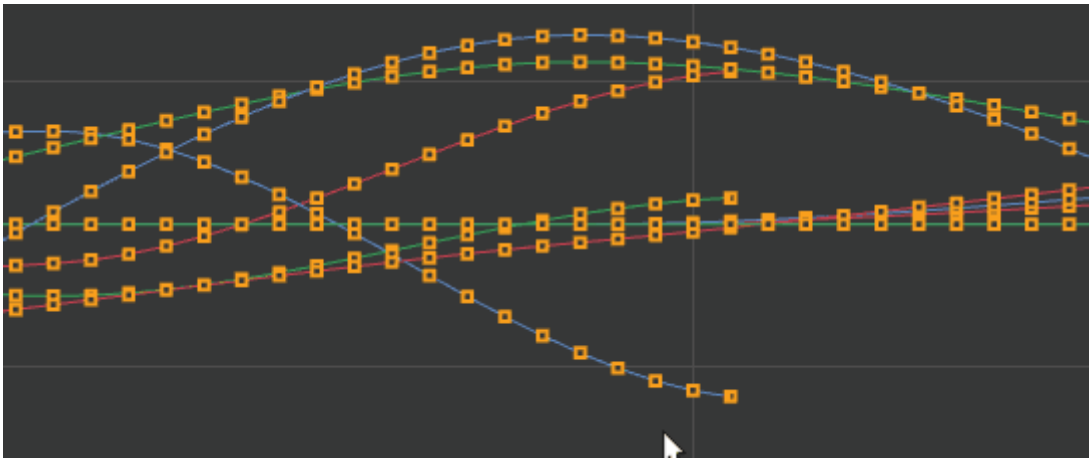




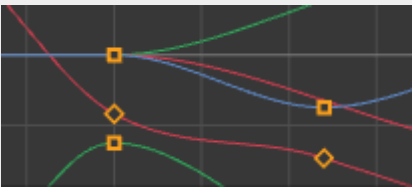
The following table provides a brief overview of tasks using the Graph Editor:

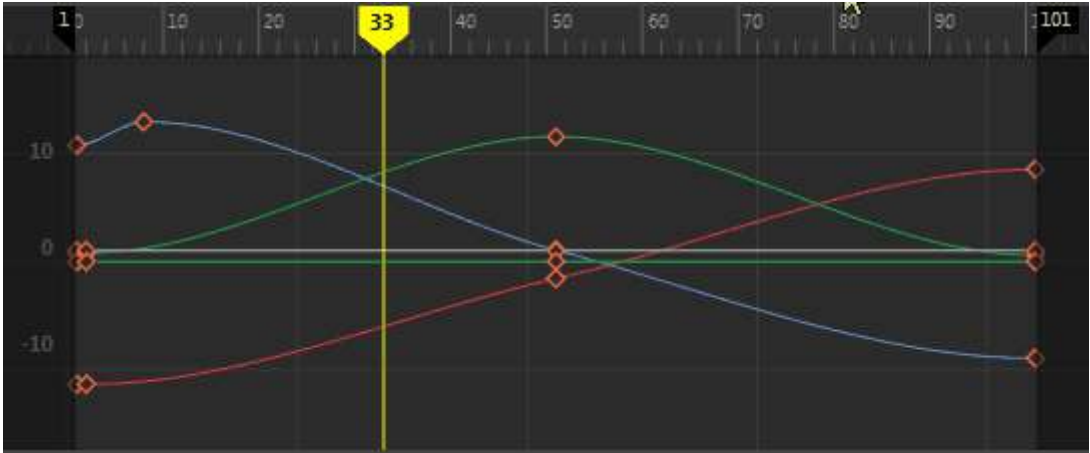
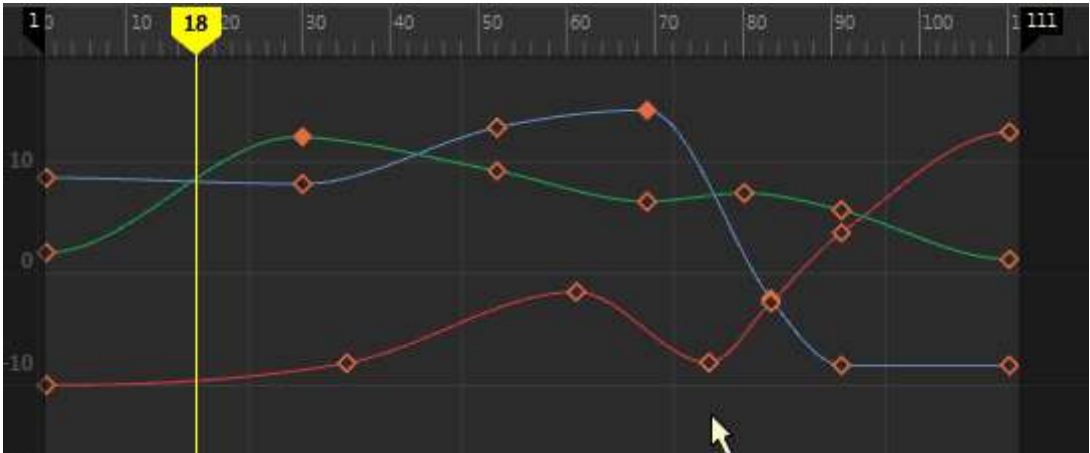
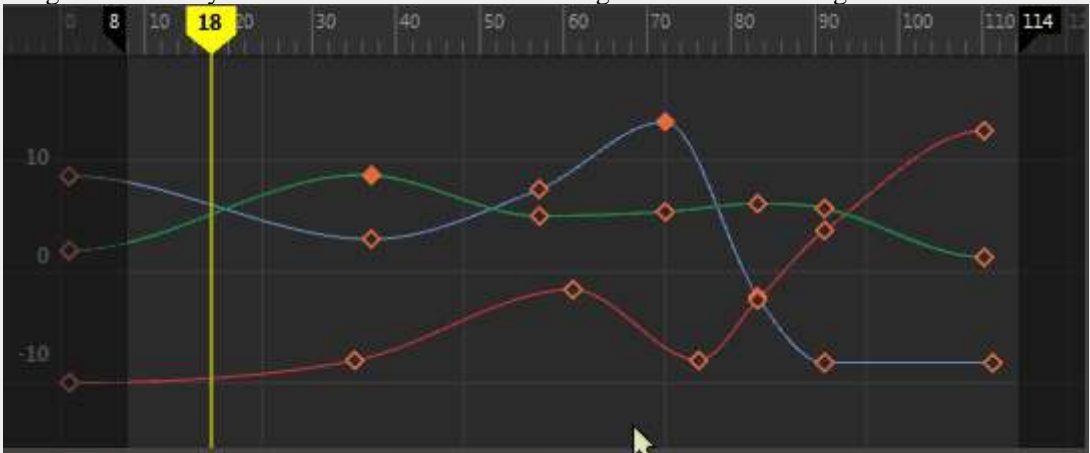
To...	Do this
Place the Graph Editor in a scene view	Select the scene view in which you want the Graph Editor to appear, then select Panels > Panel > Graph Editor.
Turn on Pre-selection highlighting in the Graph Editor	<p>In the Graph Editor menu bar, turn on Select > Pre-select Highlight.</p>  <p>Tip: Change the highlight color with the Colors window. Select Windows > Settings/Preferences > Color Settings, then change the Pre-selection highlight option in the Animation Editors section.</p>
Pan and Zoom in the Graph Editor	<p>Same as Viewport navigation:</p> <ul style="list-style-type: none"> • Pan: press Alt or Option (Mac OS X) key and middle-drag.)  <ul style="list-style-type: none"> • Zoom: press Alt or Option (Mac OS X) and drag or right-drag. Zooming is centered based on the location of the mouse cursor.

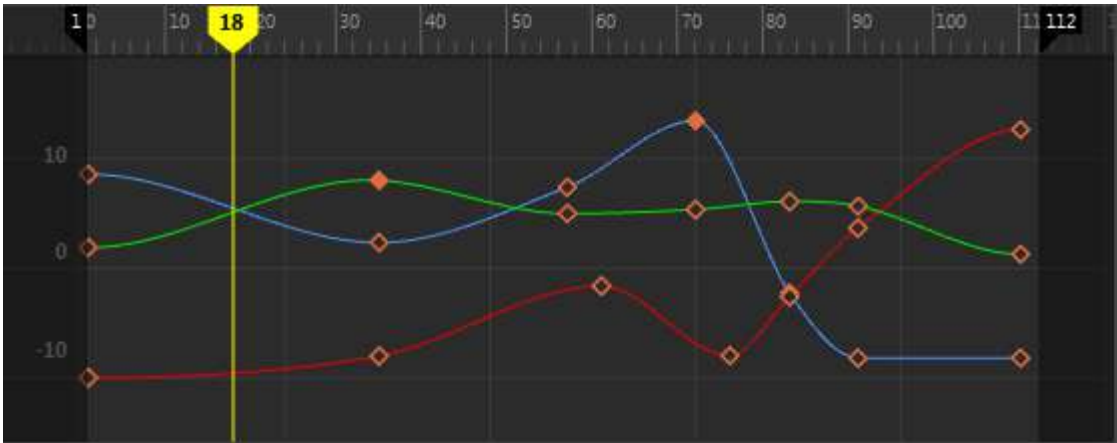
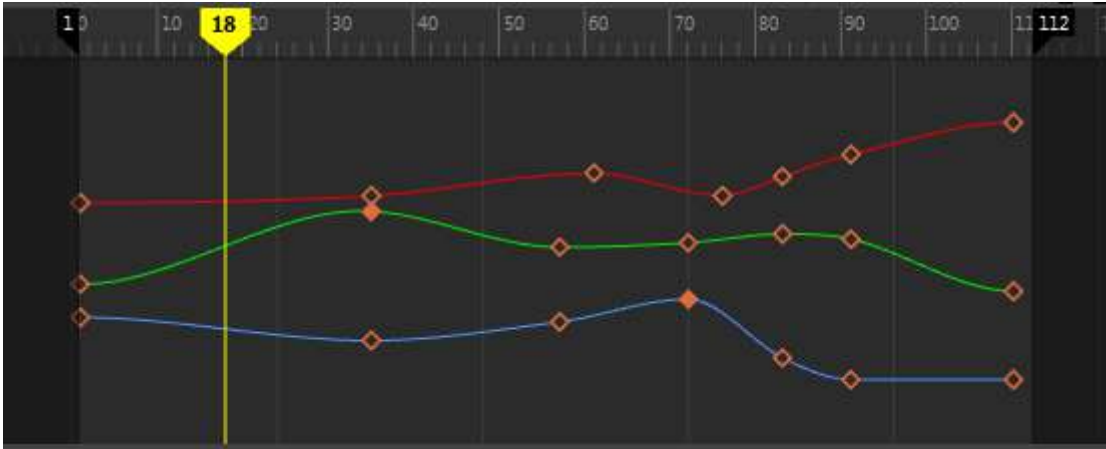
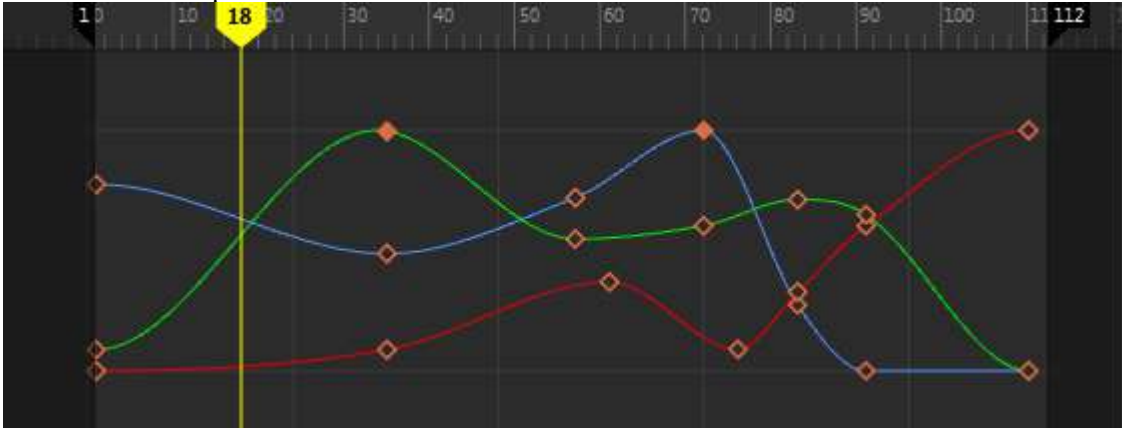
To...	Do this
	 <ul style="list-style-type: none"> Zoom horizontal: Press Alt or Option (Mac OS X) + Shift and right-drag. 
Zoom on a curve	<p>Select the animation curve (see To select a curve,below) and click F to frame the selection.</p> 
Resize the Playback range	<p>On the Time Ruler, click the edge of the dark region and drag to change the playback range. Stretch the playback range in the time view by dragging the flags on the Time ruler.</p>

To...	Do this
	
Scrub animation	<p data-bbox="293 632 649 663">Drag the Current Time Marker.</p> <ul data-bbox="391 663 1502 730" style="list-style-type: none"> <li data-bbox="391 663 1502 730">• To scrub along the Time ruler at the top of the Graph view, drag the Current Time Marker left or right.  <ul data-bbox="391 1146 1502 1247" style="list-style-type: none"> <li data-bbox="391 1146 1502 1247">• To scrub in the Graph view (below the Time ruler), press K to release the cursor lock. (The cursor is locked from scrubbing in the Time view to prevent moving keys/curves unintentionally.) 
Lock a curve	Press H to lock and J to unlock a curve.

To...	Do this
	
Move to the previous or next view	<p>Use the square bracket hotkeys ([and]). These are the same hotkeys that let you move between camera views in the viewport.</p>  <p>To navigate through views using these hotkeys, click in the Graph Editor then press left bracket ([) to move to the previous view, or right bracket (]) to move to the next view.</p>
Filter mangled Euler angles	Select the mangled animation curves (for example, Rotate X, Y, and Z), then select Curves > Euler Filter. See Graph Editor Curves menu.
Change the rotation interpolation type of existing curves	Use the Change Rotation Interp option in the Graph Editor and Dope Sheet Curves. See Graph Editor Curves menu.
Select a key	<ol style="list-style-type: none"> 1. Click the Move Nearest Picked Key Tool  on the Graph Editor Toolbar. 2. Click a key. The key becomes white to indicate it is selected. 3. Middle-click the key so the cursor becomes a crosshair . 4. Position the key.

To...	Do this
	 <p>Tip: Double-click Move Nearest Picked Key Tool  to open the options window, where you can set the Direct Key Settings. See the entry for Move Nearest Picked Key Tool in the Graph Editor Toolbar topic.</p>
Set an FK/IK key curve	Use the Set IK/FK Key option in the IK/FK Keys menu (Key > FK/IK Keys > Set IK/FK Key in the Animation menu set.)
Select a curve	<p>Curve selection is on by default.</p> <ol style="list-style-type: none"> 1. Click (or drag-select) the curve <i>between</i> two keys to select it. 2. Click (or middle-mouse-click) the curve again to move it. 
Key types	<ul style="list-style-type: none"> •  Diamond key: non-weighted tangents •  Square key: weighted tangents •  Circle key: a key on a quaternion curve (no tangents) •  Hollow key: A new key with 'automatic' tangents. The key is filled when you adjust the tangents to show that it has been edited. 
Move a key	Select a key and drag it.

To...	Do this
	 <p data-bbox="293 615 1411 648">To open the Move Key tool, select Edit > Transformation Tools > Move Keys Tool > <input type="checkbox"/> Settings.</p> <p data-bbox="293 686 1446 720">You can also middle-click an empty area of the Graph Editor and drag to reposition selected keys.</p> 
Move multiple keyframes in time without changing the attribute value	<p data-bbox="293 1224 1166 1257">Drag-select the keyframes to move. Shift-middle-drag the frames left or right.</p> 
Change views	<p data-bbox="293 1730 578 1797">Absolute view: Press 1 This view is the default.</p>

To...	Do this
	 <p>Stacked view: Press 2</p>  <p>Normalized view: press 3</p>  <p>Select View > Renormalize to revert back.</p> <p>Note: Curves are automatically renormalized when you turn on Display Normalized or the active curve list changes.</p>



EXPERIMENT 10

COMPUTER GRAPICS AND MULTIMEDIA

Aim

To bounce a ball using animation.

Syeda Reeha Quasar
14114802719
3C7

EXPERIMENT - 10

AIM:

To bounce a ball using animation.

THEORY:

The bouncing ball exercise is a good place to start, because there are not too many controllers to use in a 3D application and because it deals primarily with basic movement. You should have a good idea of what a bouncing ball looks like.

You need to set up some ground rules for animation to get started; for example, setting a frame rate for the application. You will be using 30 frames per second.

To set the frames per second or fps:

Click Window > Settings Preferences > Preferences.

In the Preferences window, select Settings.

In the Settings dialog box, set the time option to NTSC[30 fps].

To bounce a ball using Key frame Animation.

STEPS:

Step 1: Select 'Animation' from main menu bar.

Step 2: Click 'Create' on Maya title bar and then click on 'Polygon's primitives' and then click on 'Sphere'.

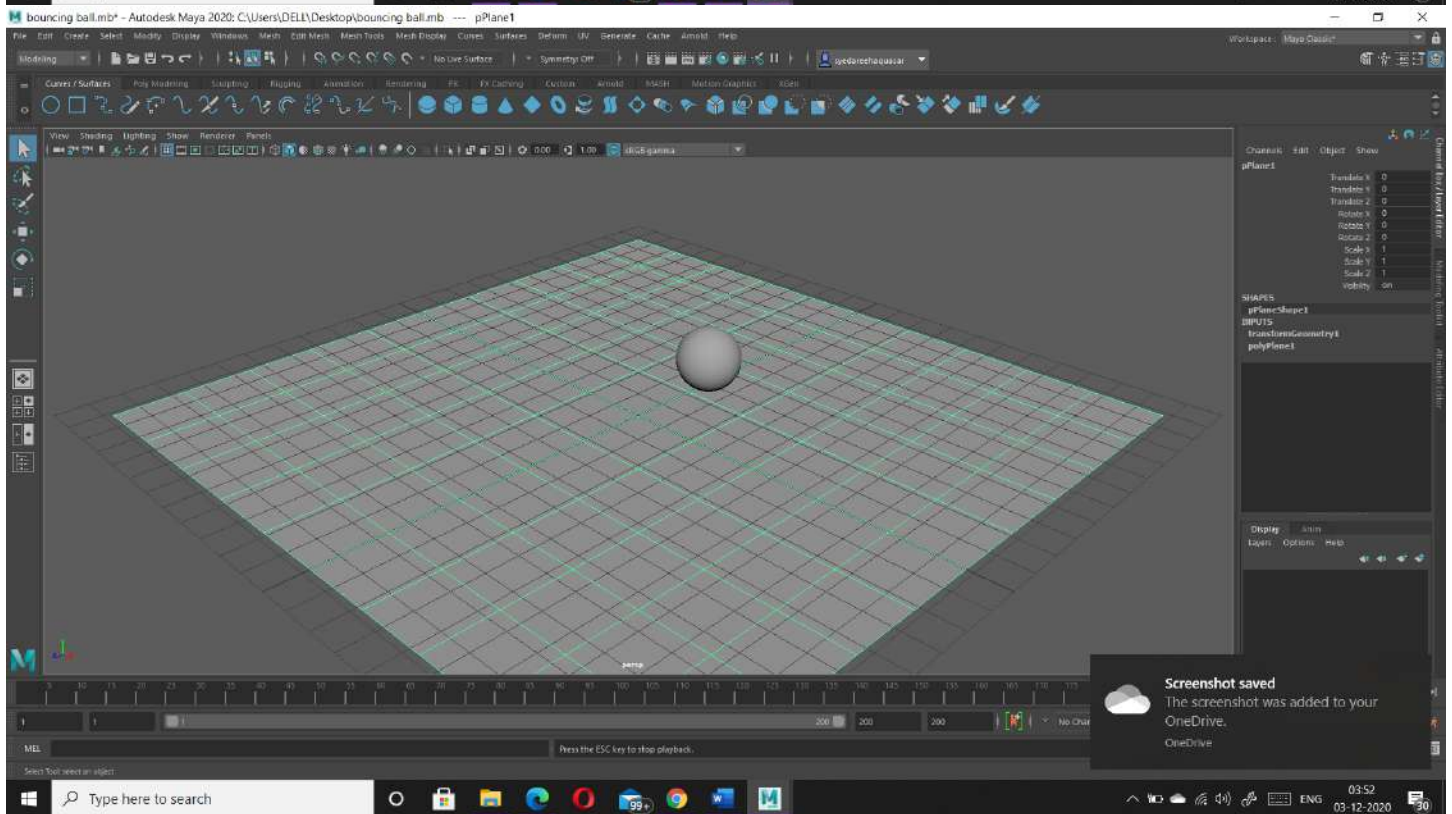
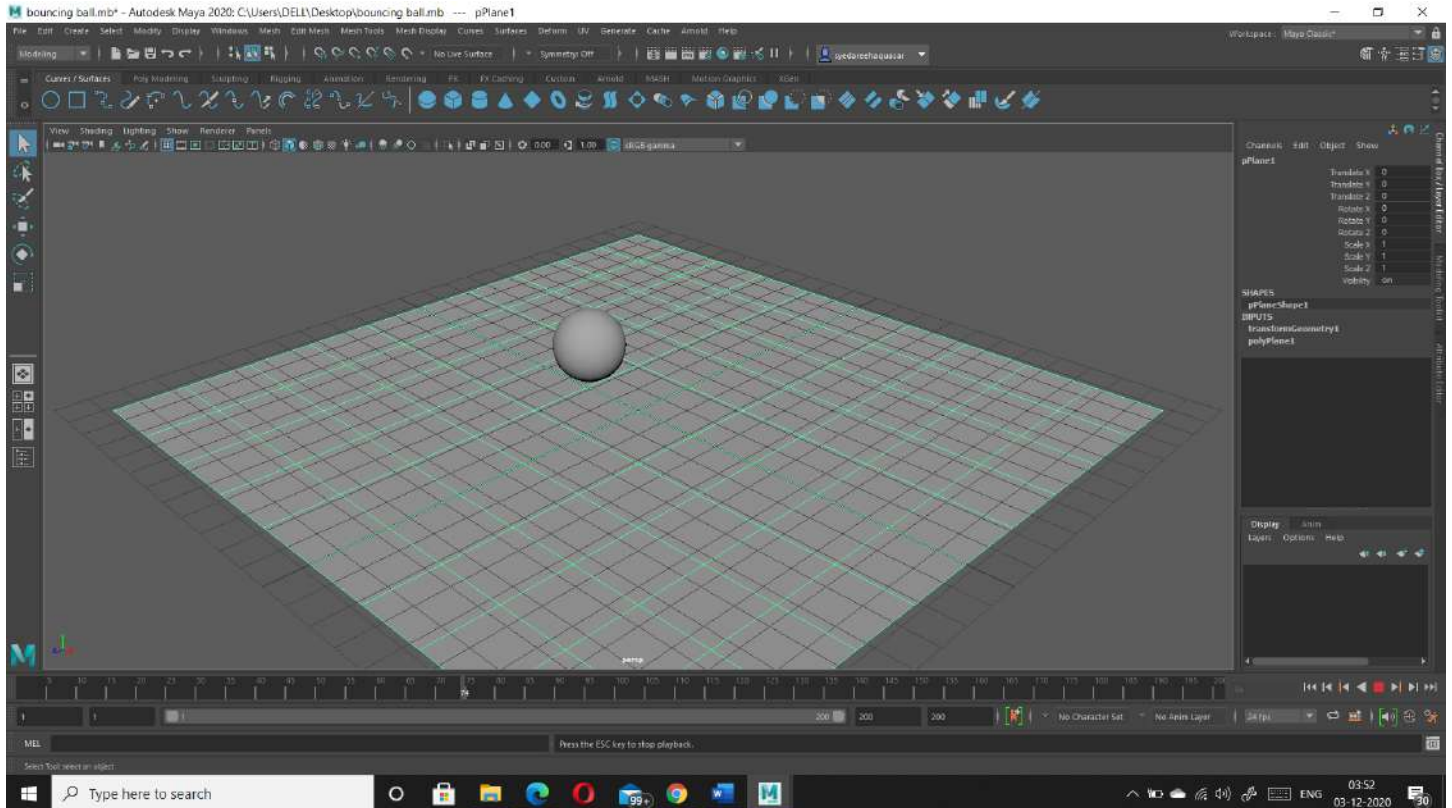
Step 3: Click 'Move' tool to move the ball to position in a frame.

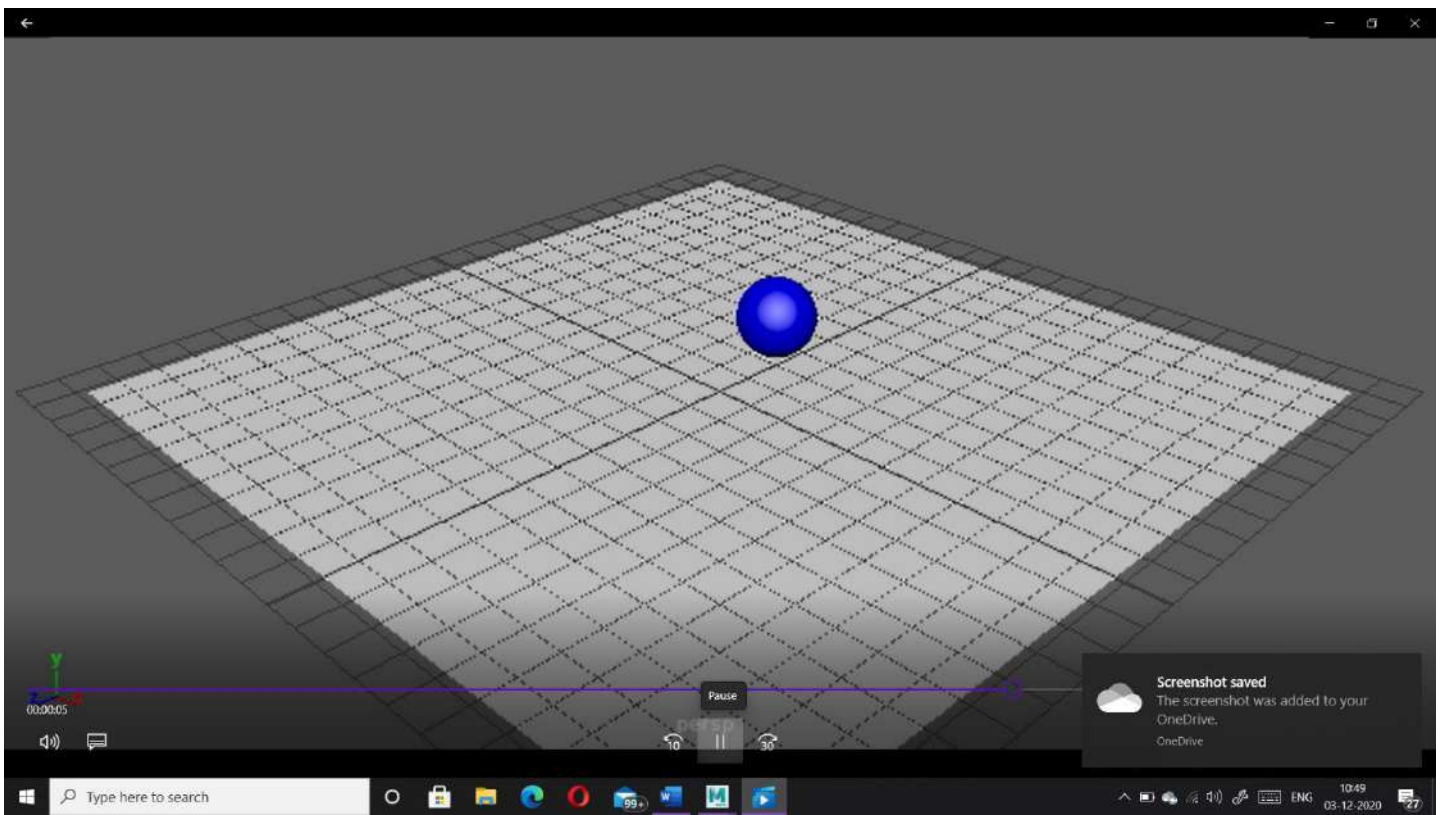
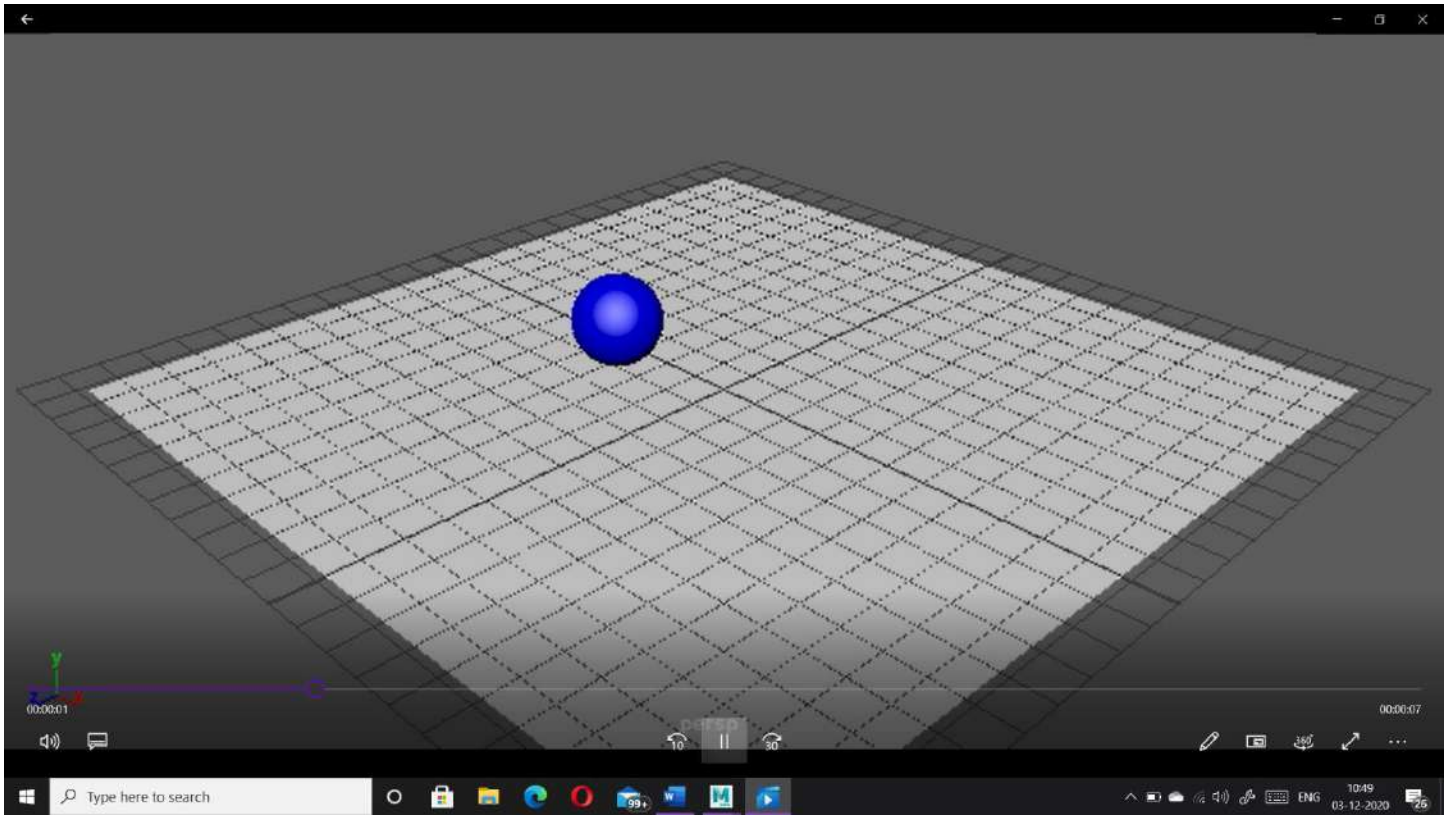
Step 4: Fix the position of the ball in a frame by pressing the 'Set key(Shortcut-S)'.

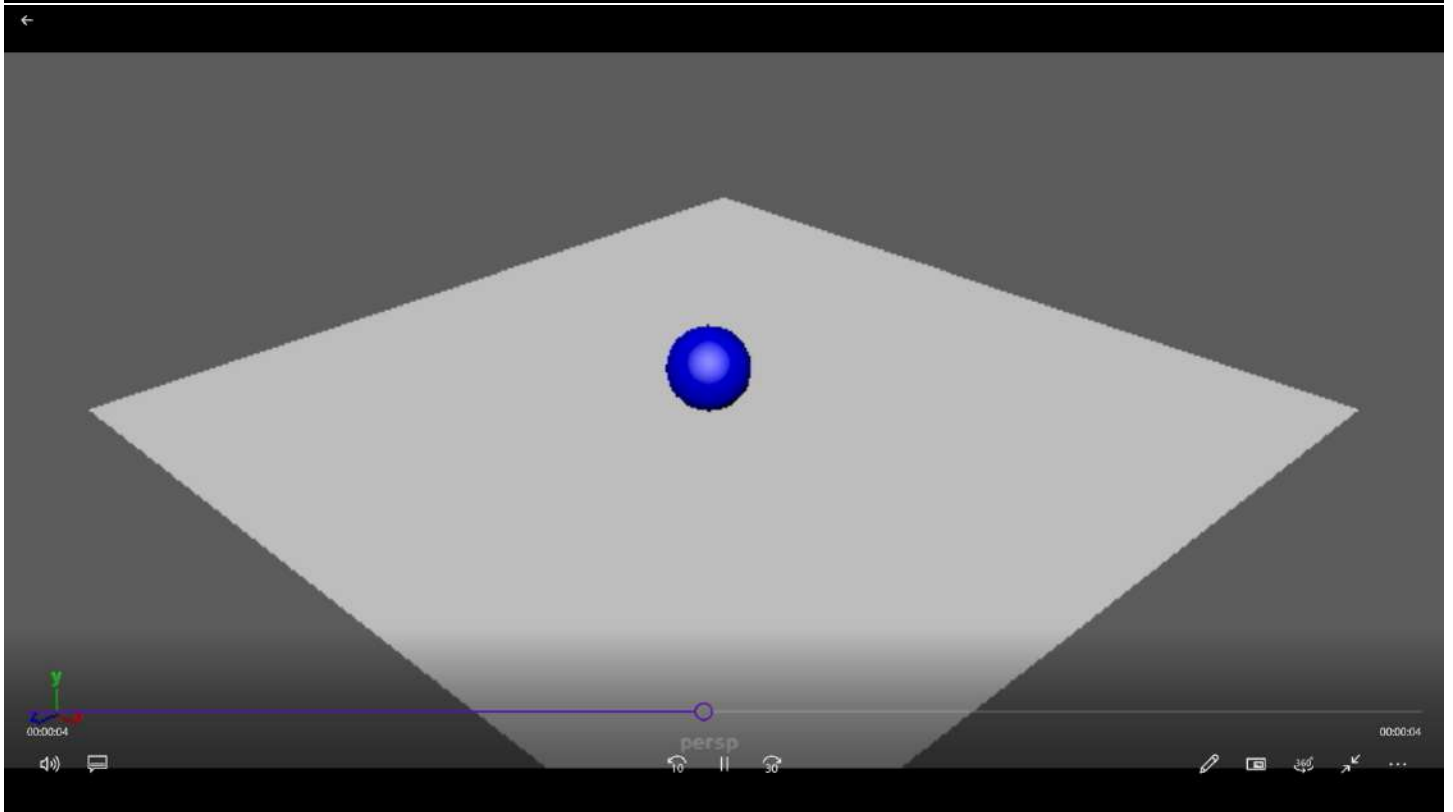
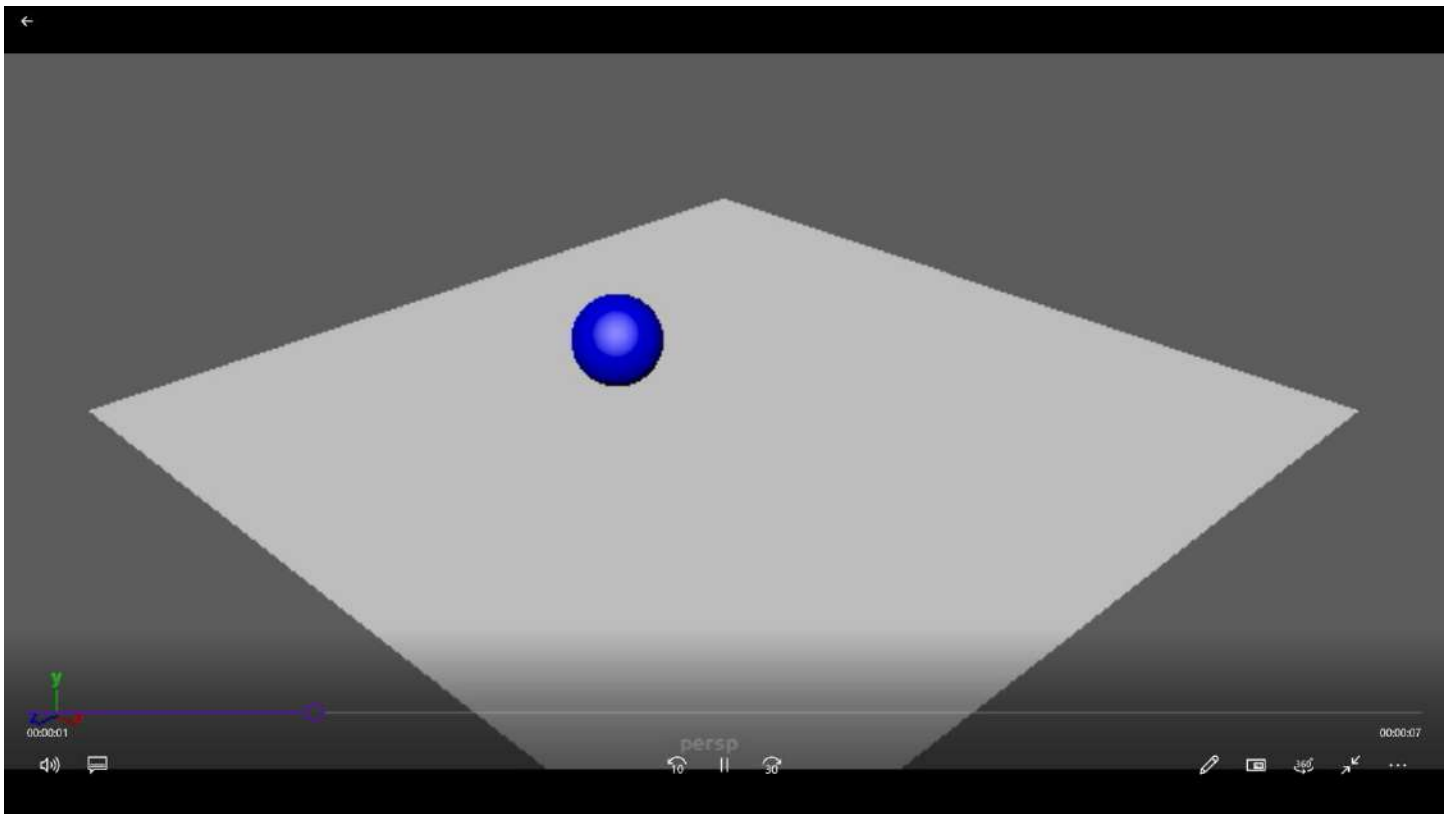
Step 5: Move to the next frame and set another position of the ball using Set key.

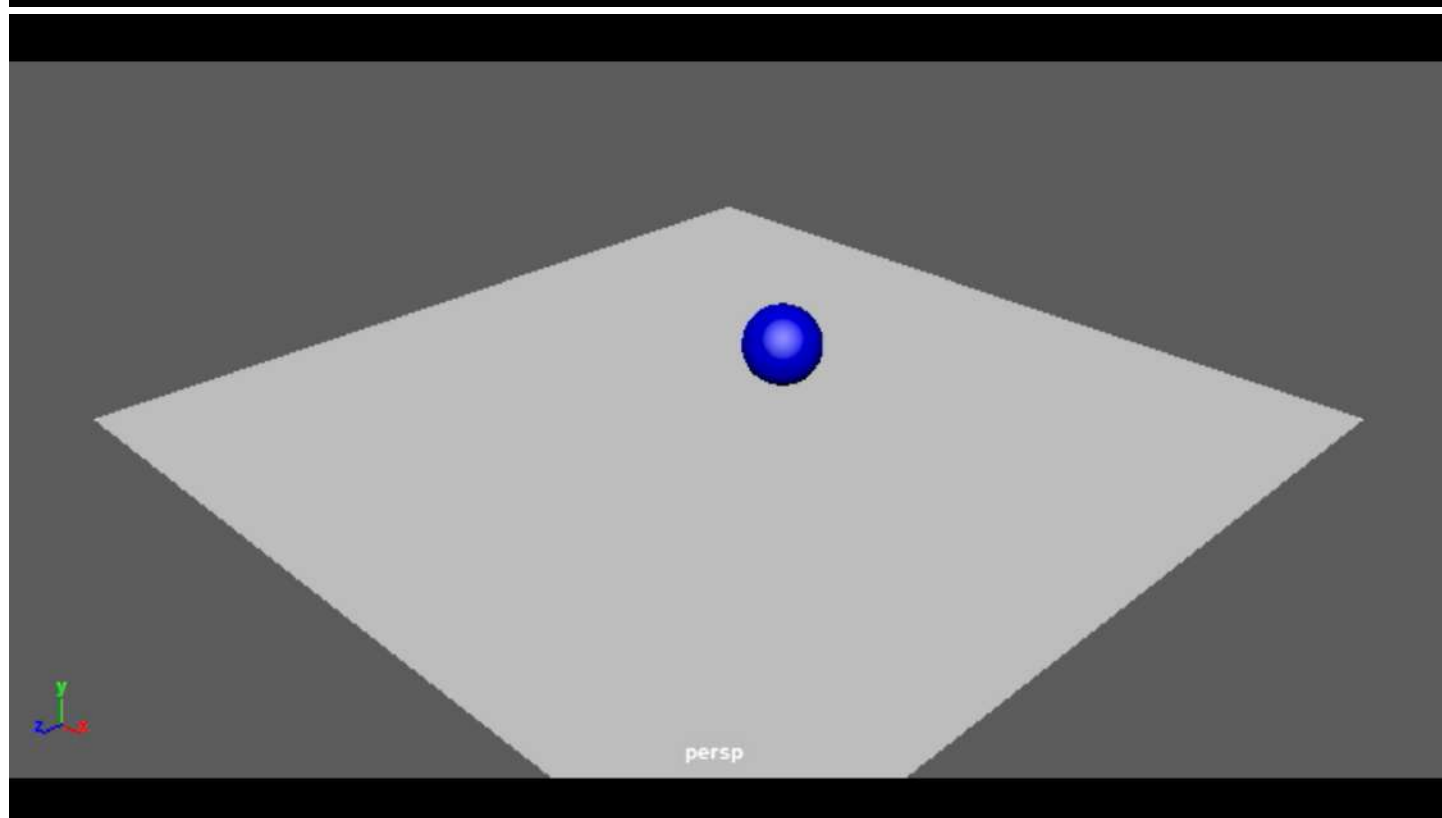
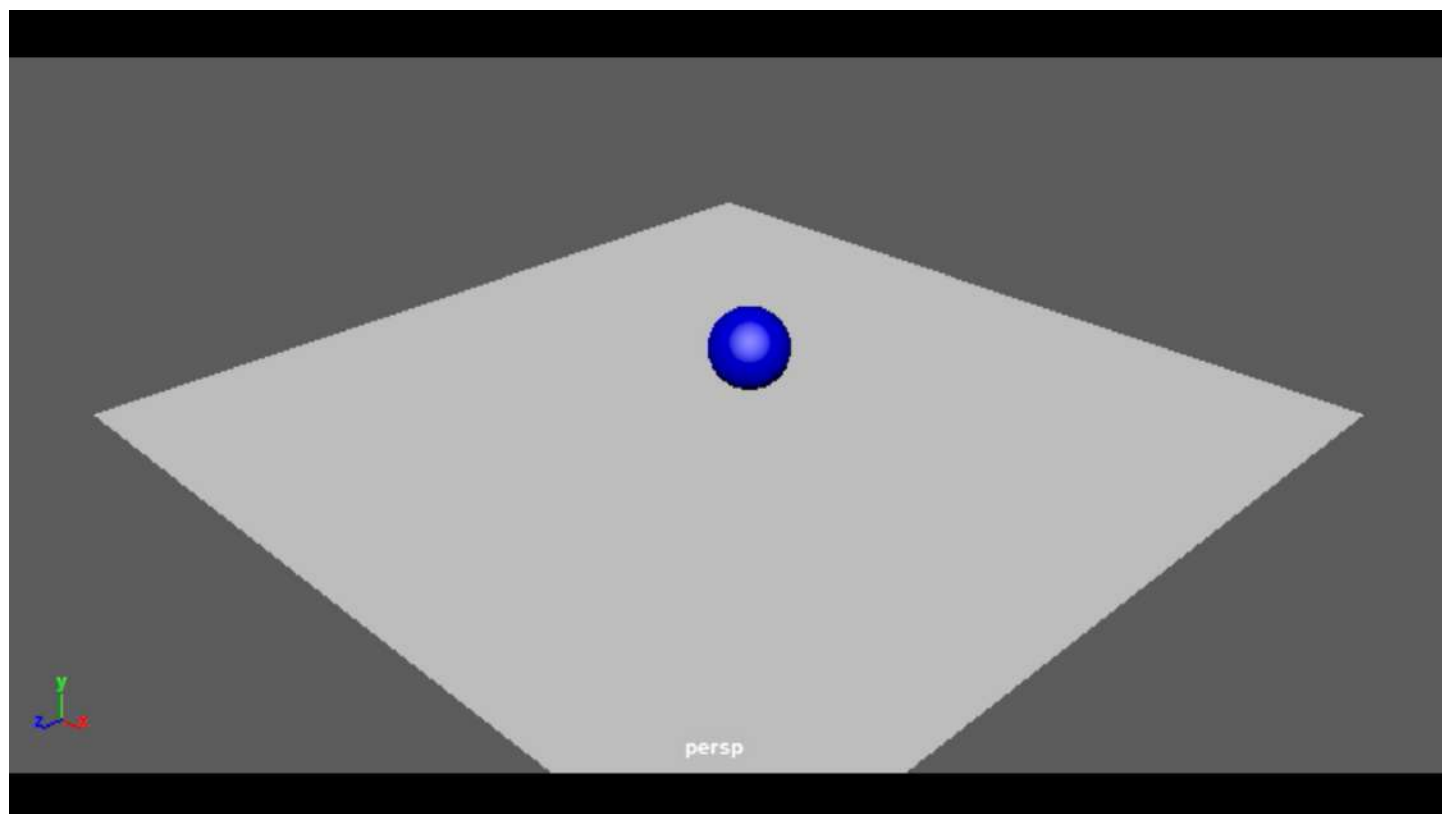
Step 6: Click 'Play' to view the moving ball.

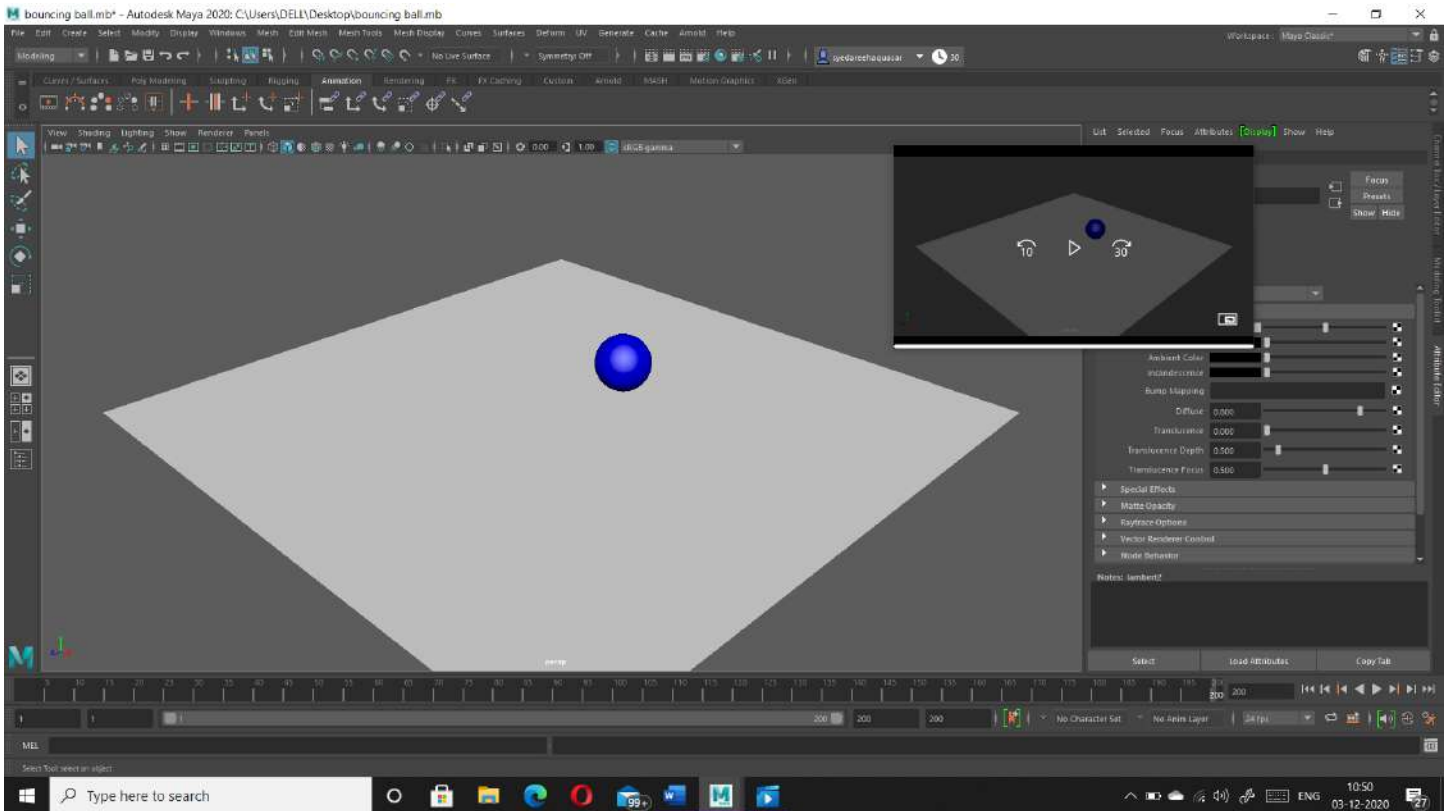
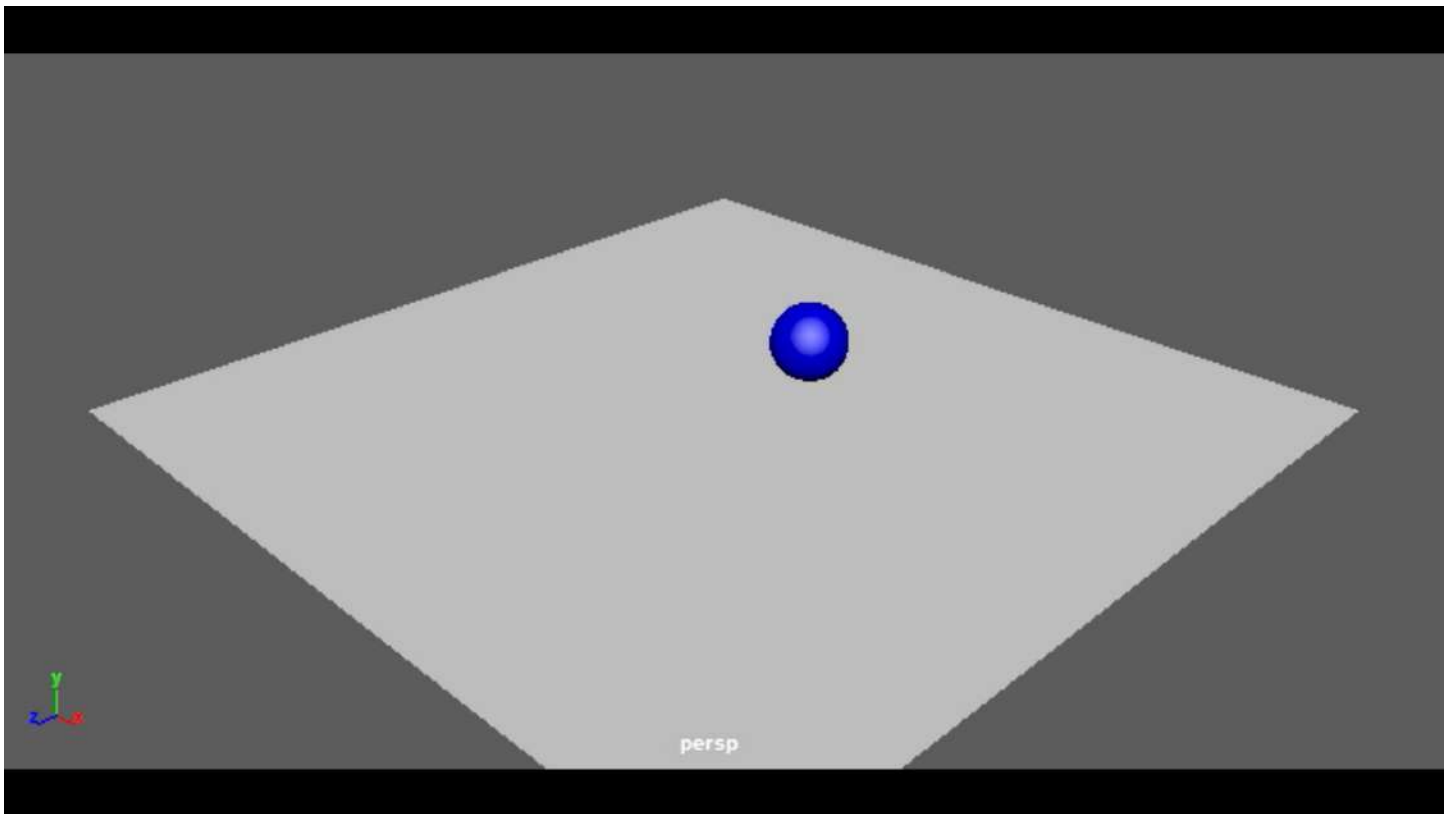
Animation Screenshots











VIVA-VOCE QUESTIONS

Q1. Short films that use stop motion techniques are what type animation?

- a) Frame-based animation
- b) HTML
- c) Animation
- d) Production

Ans.

- a) Frame-based animation

Q2. _____ is the sequence of images displayed one after the other in a given time frame.

- a) Translation
- b) Animation
- c) Ordering
- d) Shearing

Ans.

- C) Ordering

Q3 In which tab of properties window, frames per second, start time and end time is specified?

- a) Image
- b) Time
- c) Duration
- d) Animating

Ans.

- b) Time

Q4. A video consists of a sequence of

- a) Signals
- b) Frames
- c) Packets
- d) Slots

Ans.

b) Frames

Q5. A _____ is a frame in which a changes to an object's properties are defined.

a) Multiframe

b) single frame

c) keyframe

d) userframe

Ans.

c) KeyFrame



EXPERIMENT 10

COMPUTER GRAPICS AND MULTIMEDIA

Aim

Write a program to draw a car using inbuild graphics function and translate it from bottom left corner to right bottom corner of screen (Animation).

Syeda Reeha Quasar
14114802719
3C7

EXPERIMENT - 10

AIM:

Write a program to draw a car using inbuilt graphics function and translate it from bottom left corner to right bottom corner of screen (animation)

THEORY:

Making a car in computer graphics using various shapes and moving it on the screen.

Function	Description
initgraph	It initializes the graphics system by loading the passed graphics driver then changing the system into graphics mode.
getmaxx	It returns the maximum X coordinate in current graphics mode and driver.
getmaxy	It returns the maximum X coordinate in current graphics mode and driver.
setcolor	It changes the current drawing colour. Default colour is white. Each color is assigned a number, like BLACK is 0 and RED is 4. Here we are using colour constants defined inside graphics.h header file.
setfillstyle	It sets the current fill pattern and fill color.
circle	It draws a circle with radius r and centre at (x, y).
line	It draws a straight line between two points on screen.
arc	It draws a circular arc from start angle till end angle.
floodfill	It is used to fill a closed area with current fill pattern and fill color. It takes any point inside closed area and color of the boundary as input.

Function	Description
cleardevice	It clears the screen, and sets current position to (0, 0).
delay	It is used to suspend execution of a program for a M milliseconds.
closegraph	It unloads the graphics drivers and sets the screen back to text mode.

SOURCE CODE:

```
#include <stdio.h>

#include <graphics.h>

#include <conio.h>

#include <dos.h>

int main() {

    int i, x, midy;

    initwindow(800, 800);

    // get coordinates of screen

    x = getmaxx();

    midy = getmaxy()/2;

    for (i=0; i < x-150; i=i+5) {

        cleardevice();// clearing screen
```

```
setcolor(WHITE)

line(0, midy + 37, x, midy + 37); // road

    //carbody

    setcolor(BLUE);

line(i, midy + 23, i, midy);

line(i, midy, 40 + i, midy - 20);

line(40 + i, midy - 20, 80 + i, midy - 20);

line(80 + i, midy - 20, 100 + i, midy);

line(100 + i, midy, 120 + i, midy);

line(120 + i, midy, 120 + i, midy + 23);

line(0 + i, midy + 23, 18 + i, midy + 23);

arc(30 + i, midy + 23, 0, 180, 12);

line(42 + i, midy + 23, 78 + i, midy + 23);

arc(90 + i, midy + 23, 0, 180, 12);

line(102 + i, midy + 23, 120 + i, midy + 23);

line(28 + i, midy, 43 + i, midy - 15);

line(43 + i, midy - 15, 57 + i, midy - 15);

line(57 + i, midy - 15, 57 + i, midy);

line(57 + i, midy, 28 + i, midy);

line(62 + i, midy - 15, 77 + i, midy - 15);

line(77 + i, midy - 15, 92 + i, midy);

line(92 + i, midy, 62 + i, midy);
```

```
line(62 + i, midy, 62 + i, midy - 15);

//wheels

setcolor(RED);

circle(30 + i, midy + 25, 9);

circle(90 + i, midy + 25, 9);

delay(100);

}

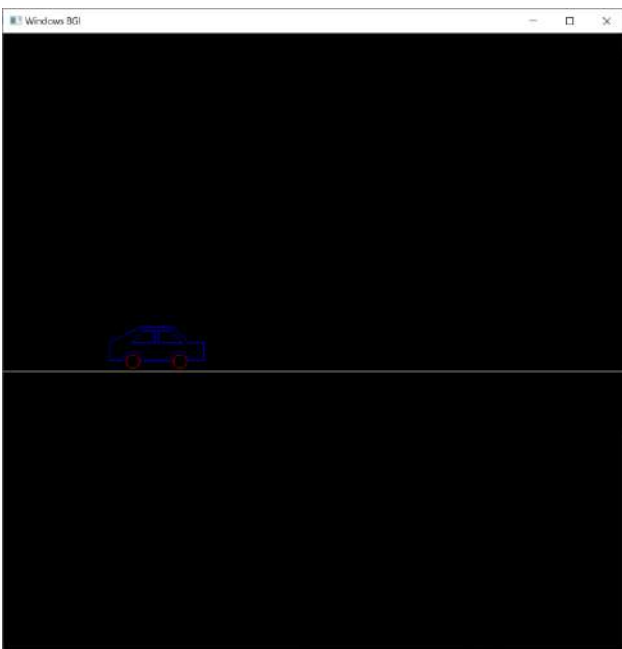
getch();

closegraph();

return 0;

}
```

OUTPUT:







EXPERIMENT 2 (ADVANCE)

COMPUTER GRAPICS AND MULTIMEDIA

Aim

Write a program to rotate a circle (alternatively inside and outside) around the circumference of another circle (animation).

Syeda Reeha Quasar
14114802719
3C7

EXPERIMENT - 2

AIM:

Write a program to rotate a circle (alternatively inside and outside) around the circumference of another circle (animation).

THEORY:

`circle(x, y, radius);`

where,

(x, y) is center of the circle.

'radius' is the Radius of the circle.

SOURCE CODE:

```
#include <graphics.h>
```

```
#include <iostream>
```

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

```
using namespace std;
```

```
void CircumRotation(int x, int y) {
```

```
    for (int angle = 0; angle < 360; angle++) {
```

```
        int nx = x + cos(angle/3.5)*100;
```

```
        int ny = y + sin(angle/3.5)*100;
```

```
        setcolor(WHITE);
```

```
        circle(nx, ny, 10);
```

```
        delay(500);
```

```
        setcolor(BLACK);
```

```
        circle(nx, ny, 10);
```

```
    }
```

```
}
```

```
void rotation(int x, int y, int radius) {
```



```

int nx = x + 120;
int ny = y;
for (int angle = 0; angle < 360; angle++) {
    setcolor(RED);
    int nx = x + cos(angle/3.5)*100;
    int ny = y + sin(angle/3.5)*100;
    circle(nx, ny, radius);
    delay(500);
    setcolor(BLACK);
    circle(nx, ny, radius);
    setcolor(YELLOW);
    nx = x + cos(angle/3.5)*120;
    ny = y + sin(angle/3.5)*120;
    circle(nx, ny, radius);
    delay(500);
    setcolor(BLACK);
    circle(nx, ny, radius);
    setcolor(BLUE);
    nx = x + cos(angle/3.5)*80;
    ny = y + sin(angle/3.5)*80;
    circle(nx, ny, radius);
    delay(500);
    setcolor(BLACK);
    circle(nx, ny, radius);
}
}

```

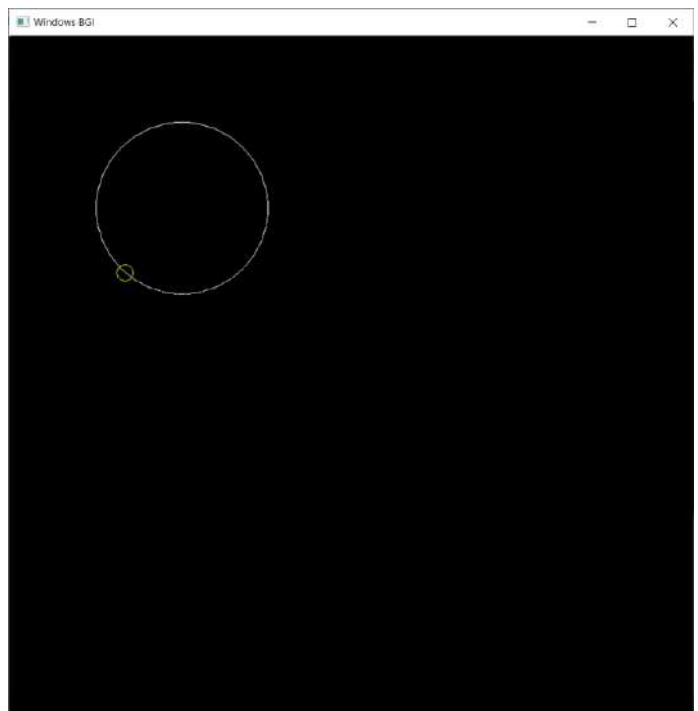
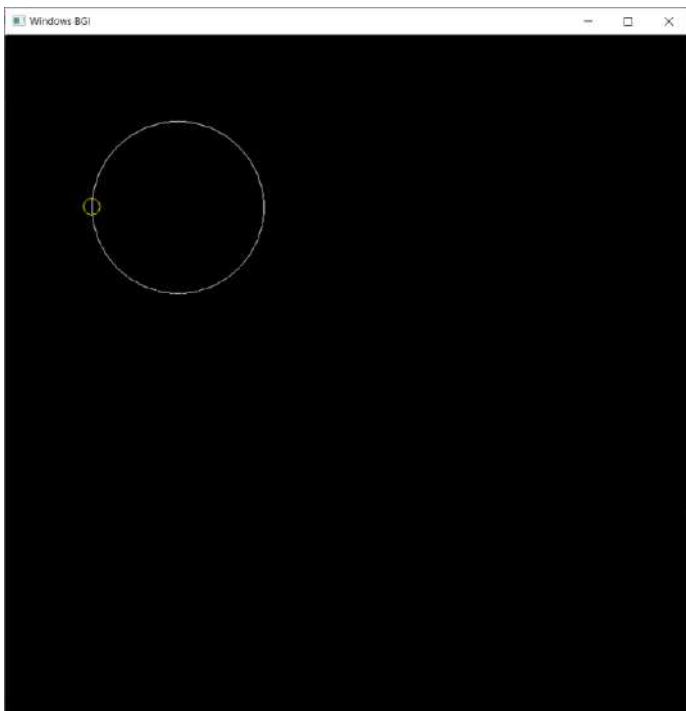
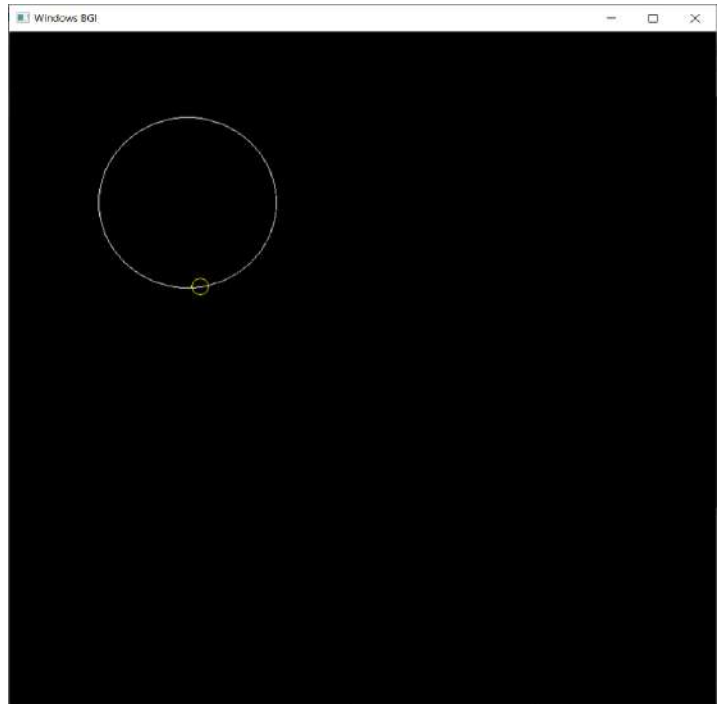
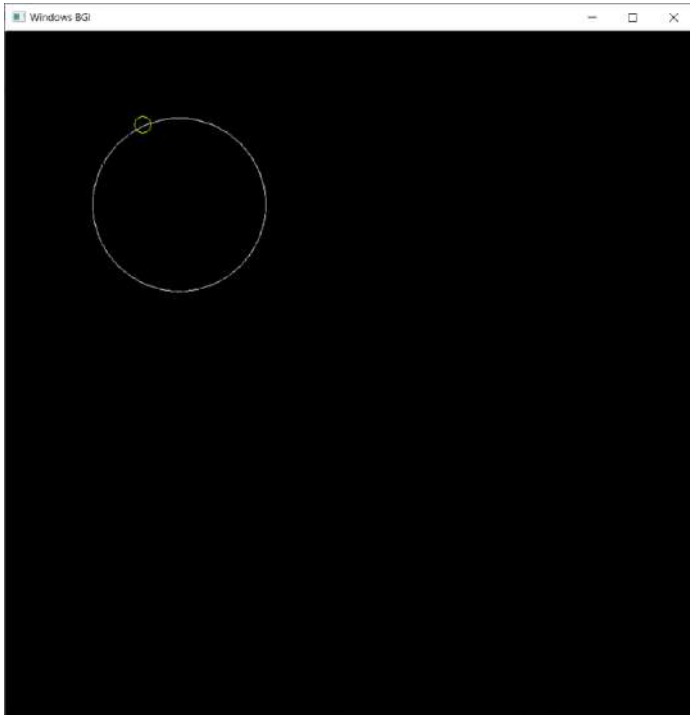
```

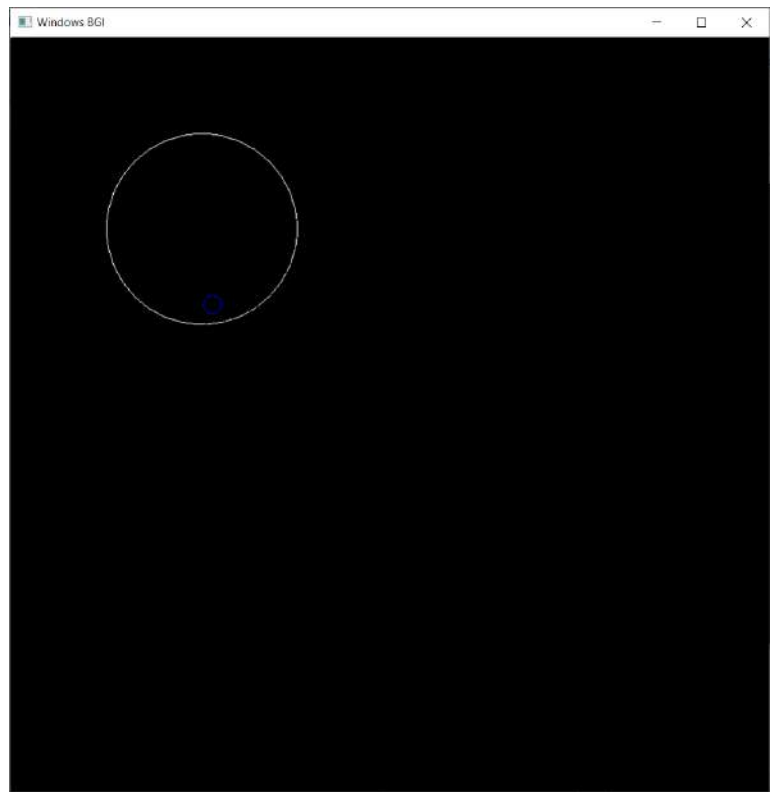
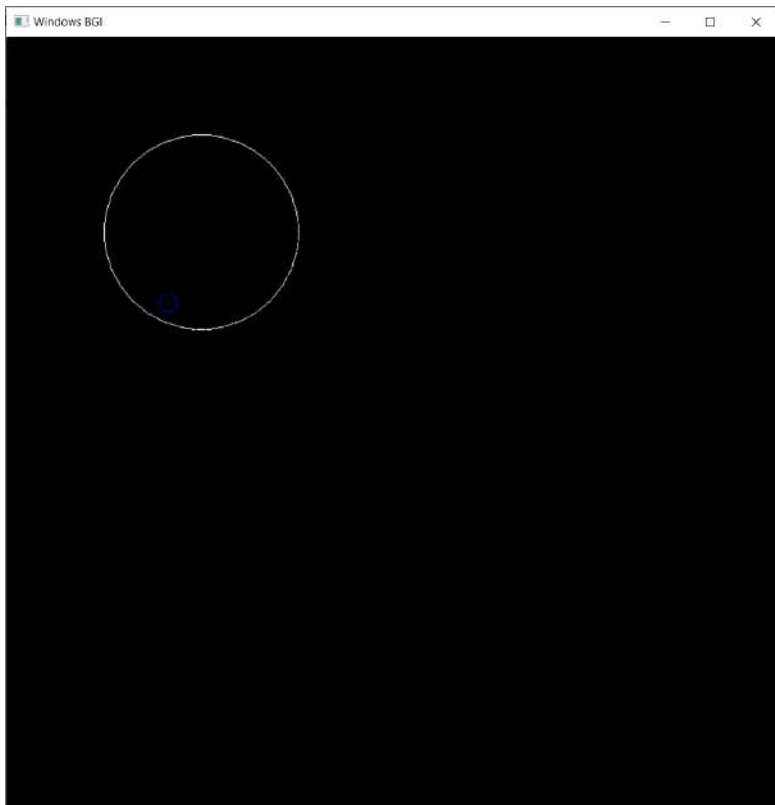
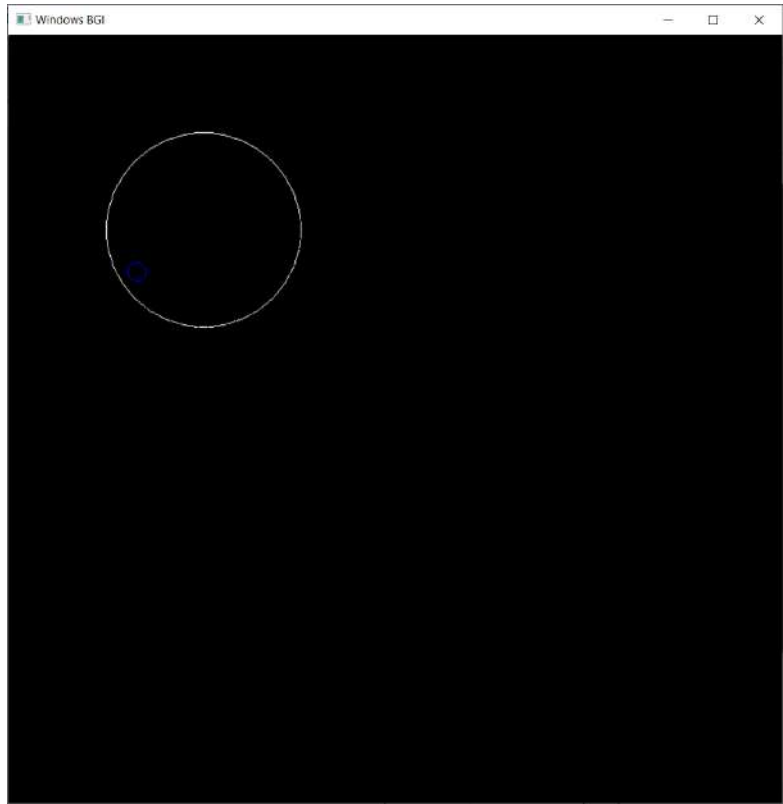
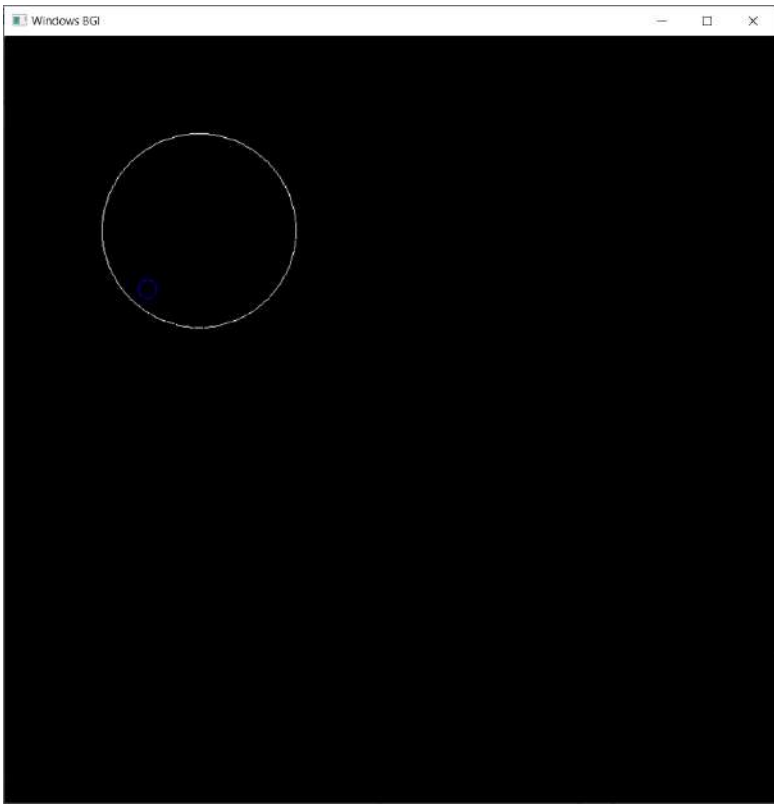
int main() {
    initwindow (800, 800);
    circle (200, 200, 100);
    // rotation(200, 200, 10);
    CircumRotation(200, 200);
}

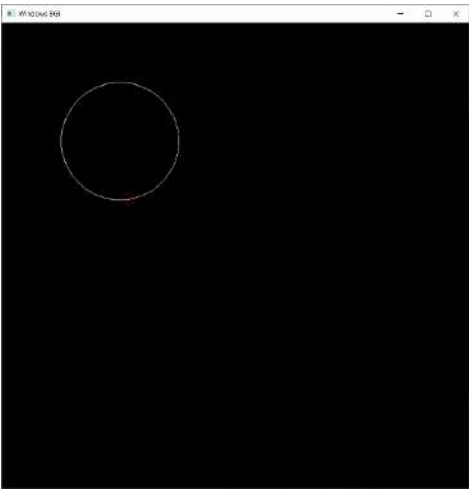
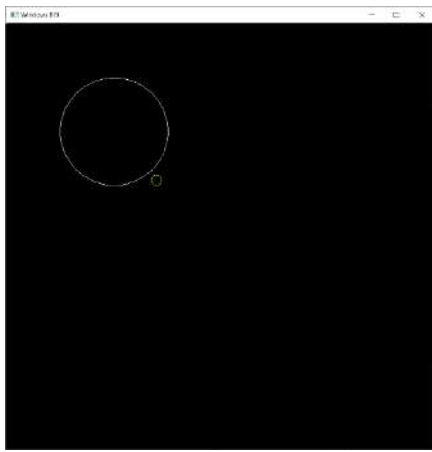
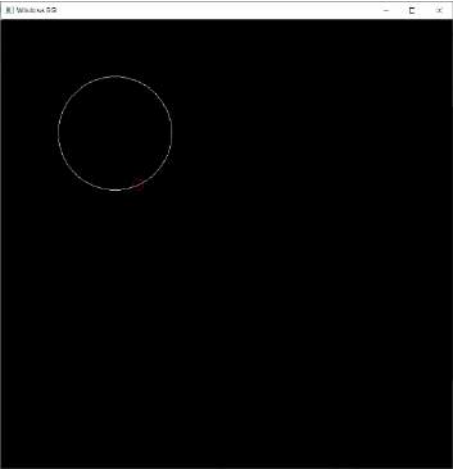
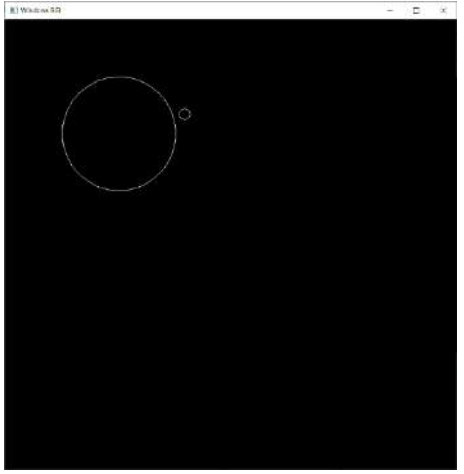
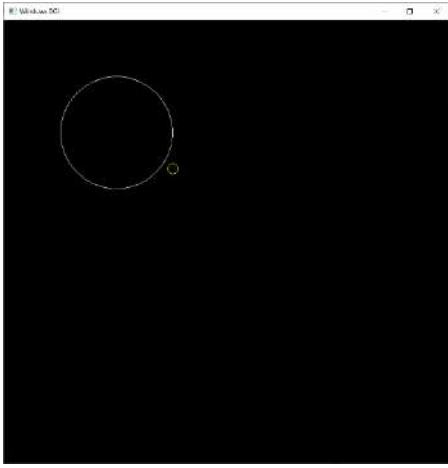
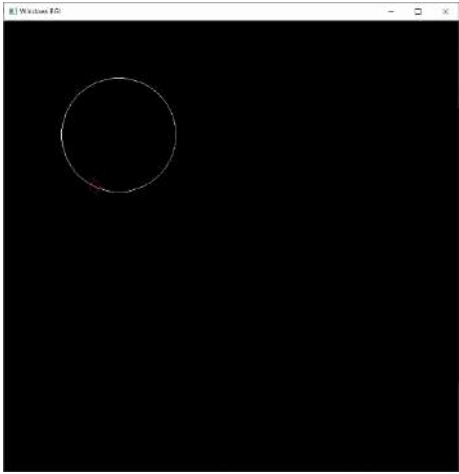
```

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

OUTPUT:









EXPERIMENT 3 BEYOND SYLLABUS

COMPUTER GRAPICS AND MULTIMEDIA

Aim

To bounce a ball using animation.

Syeda Reeha Quasar
14114802719
3C7

EXPERIMENT - 3

AIM:

To Write a program in C to draw a Rainbow.

Source Code:

```
#include <stdio.h>

#include <graphics.h>

#include <conio.h>

int main(){

    initwindow(900, 900); // window size and initialization


    //screen coordinates
    int x = getmaxx()/2;
    int y = getmaxy()/2;

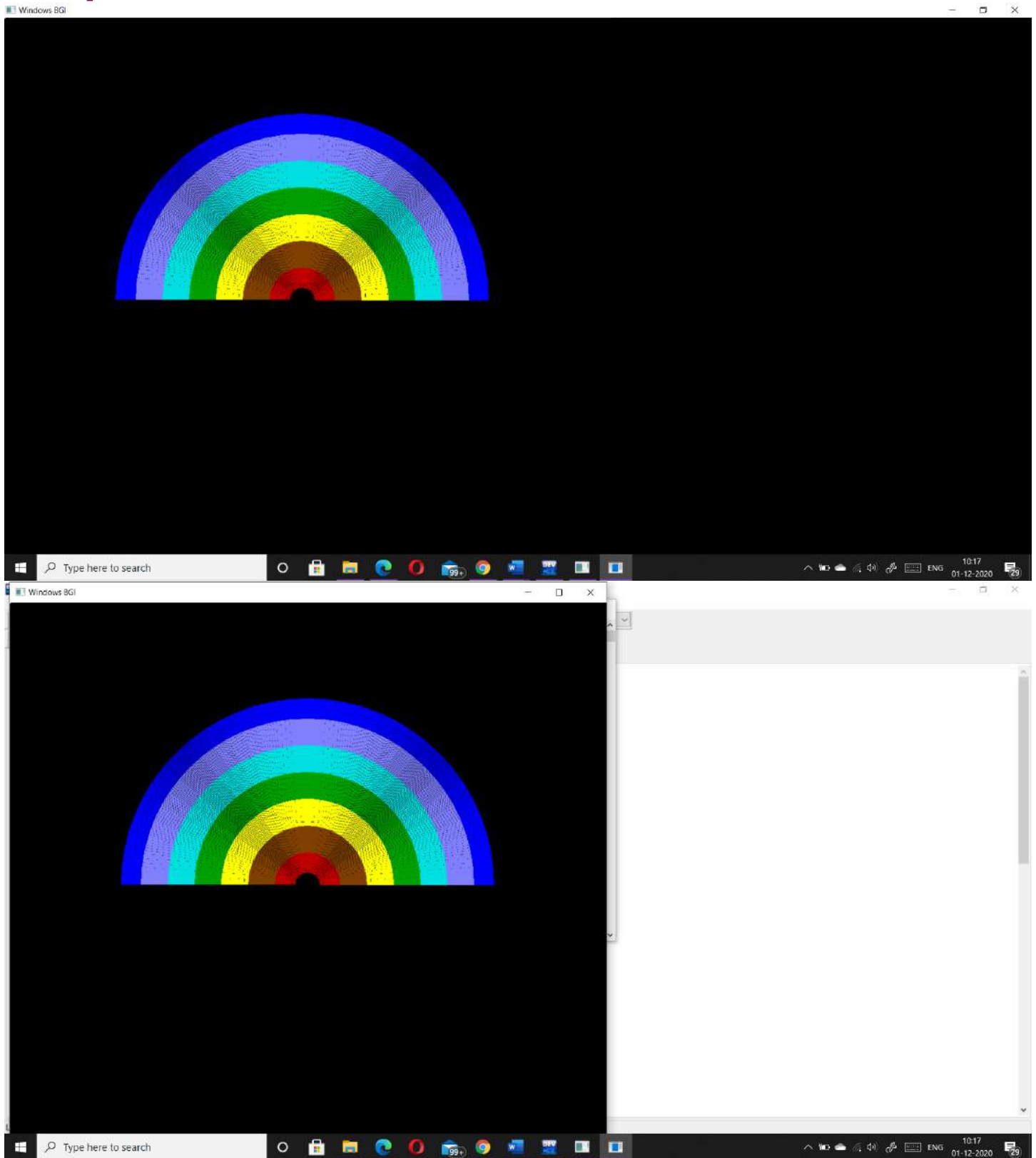

    setcolor(4); //red
    for (int i = 10; i < 40; i++) {
        arc(x, y, 0, 180, i + 10);
    }
    delay(100);


    setcolor(6); //orange (brown)
    for (int i = 40; i < 80; i++) {
        arc(x, y, 0, 180, i + 10);
    }
    delay(100);


    setcolor(14); //yellow
    for (int i = 80; i < 120; i++) {
        arc(x, y, 0, 180, i + 10);
```

```
    }  
    delay(100);  
  
    setcolor(2); // green  
    for (int i = 120; i < 160; i++) {  
        arc(x, y, 0, 180, i + 10);  
    }  
    delay(100);  
  
    setcolor(3); // aqua  
    for (int i = 160; i < 200; i++) {  
        arc(x, y, 0, 180, i + 10);  
    }  
    delay(100);  
  
    setcolor(9); // navy blue (light blue)  
    for (int i = 200; i < 240; i++) {  
        arc(x, y, 0, 180, i + 10);  
    }  
    delay(100);  
  
    setcolor(1); // indigo (blue)  
    for (int i = 240; i < 270; i++) {  
        arc(x, y, 0, 180, i + 10);  
    }  
    delay(100);  
  
    getch();  
    return 0;  
}
```


Output:





EXPERIMENT 4 BEYOND SYLLABUS

COMPUTER GRAPICS AND MULTIMEDIA

Aim

To Write a program in C to display a digital and analog clock displaying current time.

Syeda Reeha Quasar
14114802719
3C7

EXPERIMENT - 4

AIM:

To Write a program in C to display a digital and analog clock displaying current time.

Digital Clock

Source Code:

```
#include<graphics.h>

#include <time.h>

int main(){
    initwindow(1000, 500);

    time_t rawTime;
    struct tm * currentTime;
    char a[100];

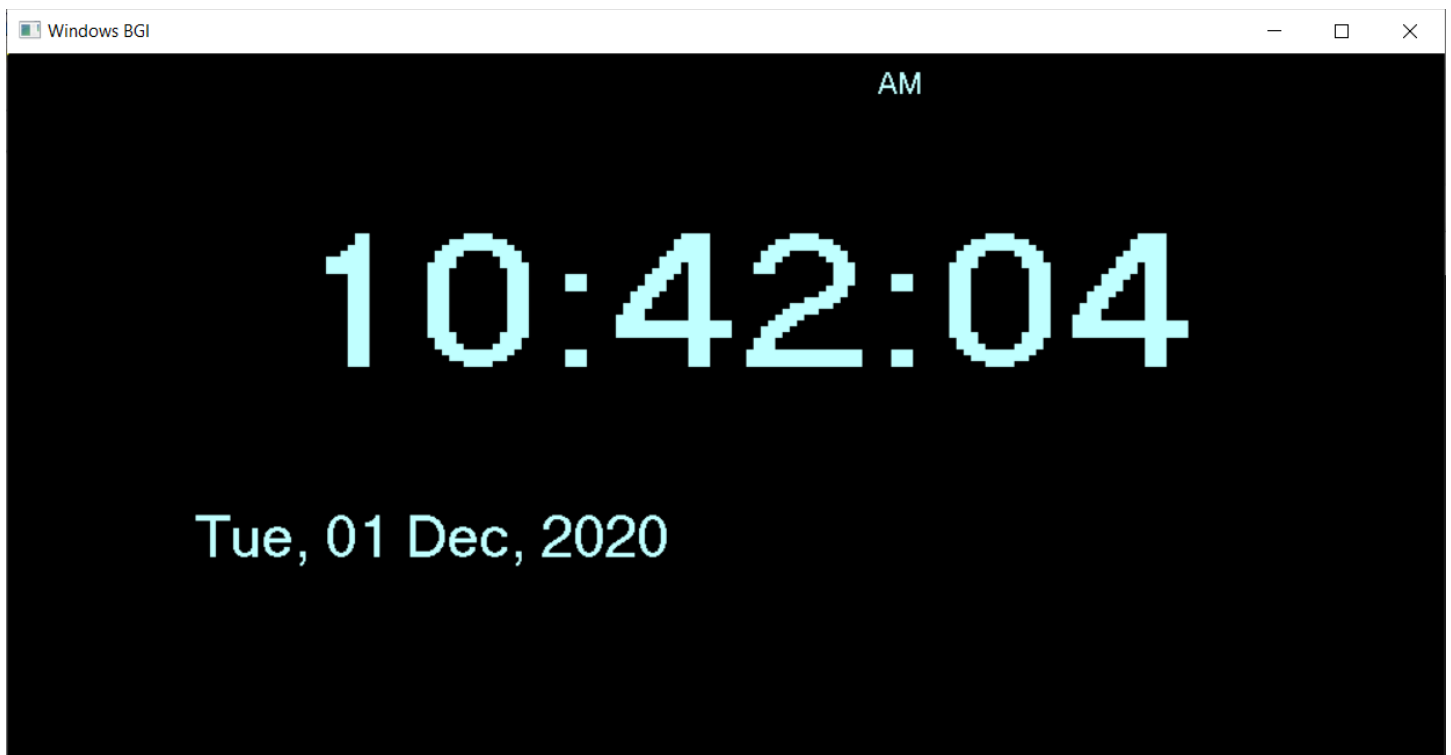
    while(1) {
        rawTime = time(NULL);
        currentTime = localtime(&rawTime);
        strftime(a, 100, "%l:%M:%S", currentTime);

        setcolor(11);
        settextstyle(3, HORIZ_DIR, 10);
        outtextxy(200, 100, a);

        strftime(a, 100, "%p", currentTime);
        settextstyle(3, HORIZ_DIR, 2);
        outtextxy(600, 8, a);
```

```
    strftime(a, 100, "%a, %d %b, %Y", currentTime);  
    settextstyle(3, HORIZ_DIR, 5);  
    outtextxy(130, 310, a);  
  
    delay(1000);  
}  
  
getch();  
return 0;  
}
```

Output:



Analog Clock

Source Code:

```
/*Program for analog CLock*/

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <string.h>
#include <graphics.h>
#include <time.h>
#include <dos.h>

void minSecPos(int xrad, int midx, int midy, int x[60], int y[60])
{
    int i, j=45;
    for (i=360; i>=0; i=i-6)
    {
        x[j] = midx-(xrad*cos((i*3.14)/180));
        y[j--] = midy-(xrad*sin((i*3.14)/180));
        j = (j== -1)?59:j;
    }
    return;
}

void calcPoints(int radius, int midx, int midy, int x[12], int y[12])
{
    int x1, y1;
    x[0] = midx, y[0] = midy-radius;
    x[6] = midx, y[6] = midy+radius;
    x[3] = midx+radius, y[3] = midy;
    x[9] = midx-radius, y[9] = midy;
```

```

x1 = (int) ((radius/2)*sqrt(3));
y1 = (radius/2);
x[2] = midx+x1, y[2] = midy-y1;
x[4] = midx+x1, y[4] = midy+y1;
x[8] = midx-x1, y[8] = midy+y1;
x[10] = midx-x1, y[10] = midy-y1;

```

```

x1 = radius/2;
y1 = (int) ((radius/2)*sqrt(3));
x[1] = midx+x1, y[1] = midy-y1;
x[5] = midx+x1, y[5] = midy+y1;
x[7] = midx-x1, y[7] = midy+y1;
x[11] = midx-x1, y[11] = midy-y1;
return;
}

```

```

int main() {
    int gd=DETECT, gm, err, tmp;
    initgraph(&gd, &gm, "C:\\tc\\bgi");

```

```

    int i, j, midx, midy, radius, hr, min, sec;
    int x[12], y[12], minx[60], miny[60];
    int hrx[12], hry[12], secx[60], secy[60];
    int secx1, secy1;
    char str[256];
    time_t t1;
    struct tm*data;

```

```

    err = graphresult();

```

```

    if (err != grOk)

```

```

{
    printf("Graphics Error: %s",
    grapherrormsg(err));
    return 0;
}

midx = getmaxx()/2;
midy = getmaxy()/2;

radius = 200;

calcPoints(radius-30, midx, midy, x, y);
calcPoints(radius-90, midx, midy, hrx, hry);

minSecPos(radius-50, midx, midy, minx, miny);
minSecPos(radius-70, midx, midy, secx, secy);

while (!kbhit())
{
    setlinestyle(SOLID_LINE, 1, 3);
    settextstyle(GOTHIC_FONT, 0, 3);

    setcolor(14);
    circle(midx, midy, radius);

    for (j=0; j<12; j++)
    {
        if (j==0)
        {
            sprintf(str, "%d", 12);
        } else {
            sprintf(str, "%d", j);
        }
    }
}

```

```
}  
setttextjustify(CENTER_TEXT, CENTER_TEXT);  
moveto(x[j], y[j]);  
outtext(str);  
}
```

```
t1 = time(NULL);  
data = localtime(&t1);
```

```
sec = data->tm_sec % 60;
```

```
setcolor(4);  
line(midx, midy, secx[sec], secy[sec]);
```

```
min = data->tm_min % 60;
```

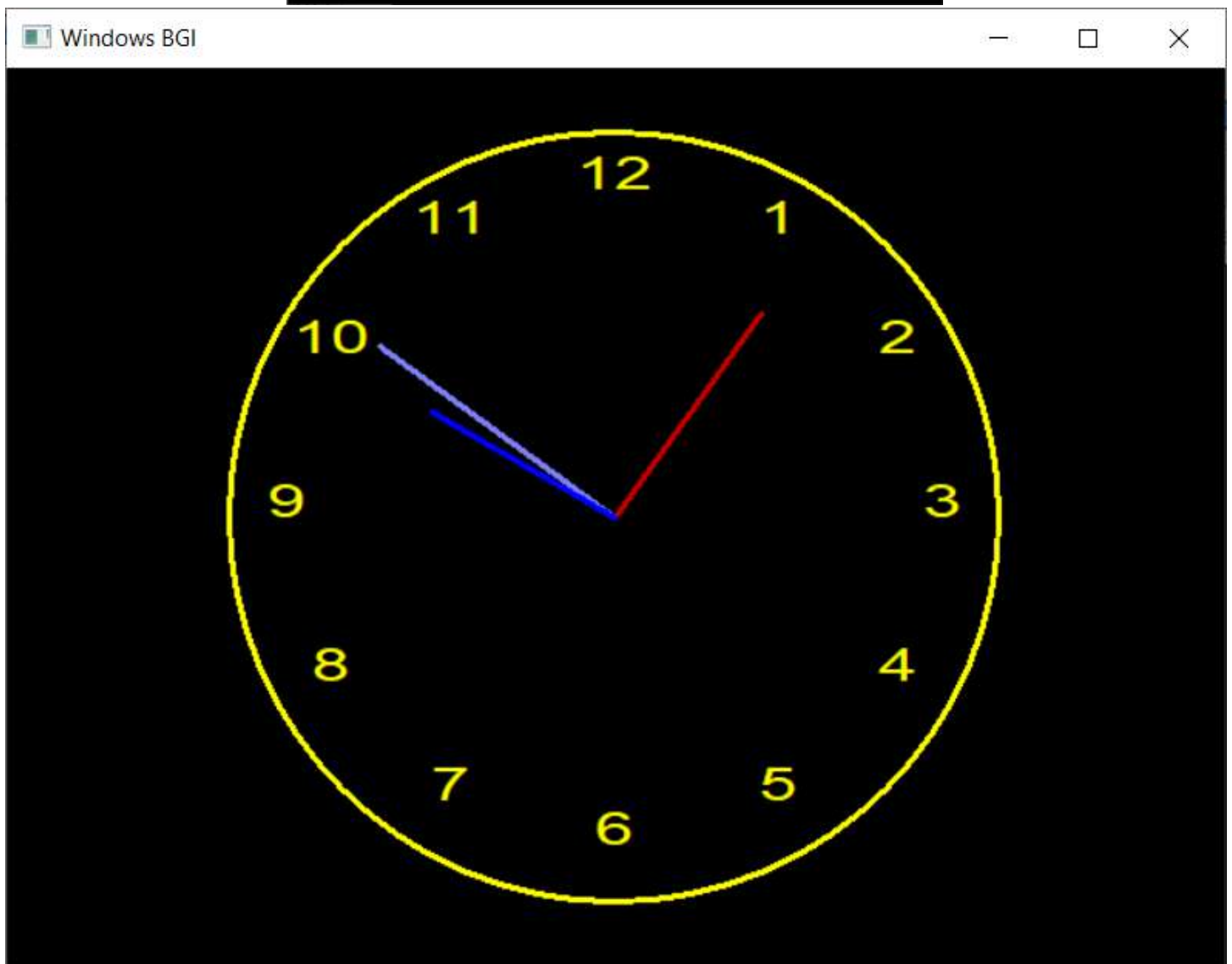
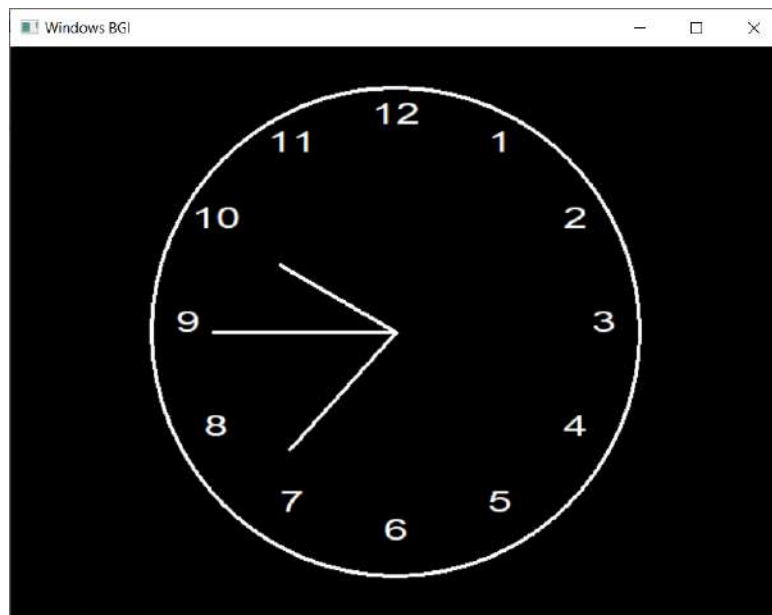
```
setcolor(9);  
line(midx, midy, minx[min], miny[min]);
```

```
hr = data->tm_hour % 12;
```

```
setcolor(1);  
line(midx, midy, hrx[hr], hry[hr]);  
delay(1000);  
cleardevice();  
}
```

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


Output:





EXPERIMENT 11

Project

COMPUTER GRAPICS AND MULTIMEDIA

Aim

Created a program using C. A man walking in Rain.

Syeda Reeha Quasar
14114802719
3C7

EXPERIMENT - 11

AIM:

Created a program using C. A man walking in Rain.

Source Code:

```
#include <stdio.h>

#include <graphics.h>

#define ScreenWidth getmaxx()
#define ScreenHeight getmaxy()
#define GroundY ScreenHeight * 0.75

int ldisp = 0;

void DrawManAndUmbrella(int x, int ldisp)
{
    //head
    circle(x, GroundY - 90, 10);
    line(x, GroundY - 80, x, GroundY - 30);
    //hand
    line(x, GroundY - 70, x + 10, GroundY - 60);
    line(x, GroundY - 65, x + 10, GroundY - 55);
    line(x + 10, GroundY - 60, x + 20, GroundY - 70);
    line(x + 10, GroundY - 55, x + 20, GroundY - 70);
    //legs
    line(x, GroundY - 30, x + ldisp, GroundY);
    line(x, GroundY - 30, x - ldisp, GroundY);
    //umbrella
    pieslice(x + 20, GroundY - 120, 0, 180, 40);
    line(x + 20, GroundY - 120, x + 20, GroundY - 70);
```

```
}
```

```
void Rain(int x)
```

```
{
```

```
    int i, rx, ry;
```

```
    for (i = 0; i < 400; i++)
```

```
    {
```

```
        rx = rand() % ScreenWidth;
```

```
        ry = rand() % ScreenHeight;
```

```
        if (ry < GroundY - 4)
```

```
        {
```

```
            if (ry < GroundY - 120 || (ry > GroundY - 120 && (rx < x - 20 || rx > x + 60)))
```

```
                line(rx, ry, rx + 0.5, ry + 4);
```

```
        }
```

```
    }
```

```
}
```

```
int main(void)
```

```
{
```

```
    int x = 0;
```

```
    //Change BGI directory according to yours
```

```
    initwindow(800, 800);
```

```
    while (!kbhit())
```

```
    {
```

```
        //Draw Ground
```

```
        line(0, GroundY, ScreenWidth, GroundY);
```

```
        Rain(x);
```

```
        ldisp = (ldisp + 2) % 20;
```

```
        DrawManAndUmbrella(x, ldisp);
```

```
        delay(75);
```

```
        cleardevice();
```

```
        x = (x + 2) % ScreenWidth;
```

```
    }
```

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

OUTPUT







