SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203

DEPARTMENT

OF

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

ACADEMIC YEAR: 2025-2026 (ODD SEMESTER)

LAB MANUAL

(REGULATION - 2023)

1922707 – VIRTUAL REALITY LABORATORY

SEVENTH SEMSTER

B.TECH – ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Prepared By

Dr.R.THENMOZHI, Asso.Prof. / AI&DS

VISION OF THE DEPARTMENT

To become a model for Artificial Intelligence with innovation and analysis for higher learning through various analytical knowledge, creative competent and dynamic technocrats; while remaining responsive to ethical, societal and environmental issues.

MISSION OF THE DEPARTMENT

M1: To develop the students as an Artificial Intelligence designer and data analyst professionals in order to meet the global design challenges and entrepreneurs of International excellence as global leaders capable of contributing towards technological innovations, learning process, participation citizenship in their neighbourhood and economic growth.

M2: To transform value-based data science education to the students and groom them as leaders in the field of Artificial Intelligence and Data Science for the empowerment of society.

INDEX

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

- To afford the necessary background in the field of Artificial Intelligence and data Science to deal with engineering problems to excel as engineering professionals in industries.
- To improve the qualities like creativity, leadership, teamwork and skill thus contributing towards the growth and development of society
- To develop ability among students towards innovation and entrepreneurship that caters to the needs of Industry and society.
- To inculcate and attitude for life-long learning process through the use of Artificial Intelligence and Data Science sources.
- To prepare then to be innovative and ethical leaders, both in their chosen profession and in other activities.

PROGRAMME OUTCOMES (POs)

After going through the four years of study, Information Technology Graduates will exhibit ability to:

PO #	Graduate Attribute	Program Outcome
1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2	Problem analysis	Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	Design/development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions
5	Modern tool usage	Create, select, and apply appropriate techniques, resources, and

		modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
6	The engineer and society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7	Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings
10	Communication	Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions
11	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
12	Life-long learning	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

PROGRAMME SPECIFIC OUTCOMES (PSOs)

After the completion of Bachelor of Technology in Artificial Intelligence and Data Science programme the student will have following Program specific outcomes.

- Design and develop secured database applications with data analytical approaches of data preprocessing, optimization, visualization techniques and maintenance using state of the art methodologies based on ethical values.
- Design and develop intelligent systems using computational principles, methods and systems for extracting knowledge from data to solve real time problems using advanced technologies and tools.
- Design, plan and setting up the network that is helpful for contemporary business environments using latest software and hardware.
- Planning and defining test activities by preparing test cases that can predict and correct errors ensuring a socially transformed product catering all technological needs.

1922707 **VIRTUAL REALITY LABORATORY** **L T P C**

0 0 4 2

OBJECTIVES:

1. To make the students understand graphics programming
2. To perform 2D and 3D transformation
3. To create 2D animations
4. To create a 3D scenes
5. To create VR architecture using mobile and WebVR/XR

LIST OF EXPERIMENTS:

1. Implementation of Algorithms for drawing 2D Primitives – Line using DDA and Bresenham's algorithm
2. Implementation of midpoint circle generating algorithm
3. 2D Geometric transformations – Translation, Rotation, Scaling, Reflection, Shear
4. Implementation of Line Clipping Algorithm
5. 3D Transformations - Translation, Rotation, Scaling
6. Creating 3D Scenes
7. 2D Animation – To create Interactive animation using any authoring tool
8. Hand held VR mobile app development using unity
9. 3D modelling for VR
10. VR walkthrough (mobile app & WebVR /XR)
11. VR Game development (using unity/Blender)

Total: 60 Periods

OUTCOMES:

At the end of the course, the student should be able to:

1. Understand graphics programming
2. Design two and three dimensional graphics and apply transformations
3. Apply clipping techniques to graphics and To create 2D animations and 3D scenes
4. To create VR architecture using mobile and WebVR/XR
5. To create simple VR Game

LIST OF EQUIPMENTS FOR A BATCH OF 30 STUDENTS

SOFTWARE: C/OPENGL/Blender/Unity

HARDWARE: Standalone desktops 30 Nos

C programming is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system. C is the most widely used computer language. It keeps fluctuating at number one scale of popularity along with Java programming language, which is also equally popular and most widely used among modern software programmers.

- The UNIX OS was totally written in C.
- Today C is the most widely used and popular System Programming Language.
- Most of the state-of-the-art software have been implemented using C.
- Today's most popular Linux OS and RDBMS MySQL have been written in C.

Turbo C

Turbo C was an integrated development environment (IDE) for programming in the C language. It was developed by Borland and first introduced in 1987. At the time, Turbo C was known for its compact size, comprehensive manual, fast compile speed and low price.

Blender

Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. Blender is well suited to individuals and small studios who benefit from its unified pipeline and responsive development process. Examples from many Blender-based projects are available in the showcase.

Blender is cross-platform and runs equally well on Linux, Windows, and Macintosh computers. Its interface uses OpenGL to provide a consistent experience. To confirm specific compatibility, the list of supported platforms indicates those regularly tested by the development team. Blender has no price tag, but you can invest, participate, and help to advance a powerful collaborative tool: Blender is your own 3D software.

COURSE OUTCOMES

1908402.1	Understand graphics programming
1908402.2	Design two and three dimensional graphics and apply transformations
1908402.3	Apply clipping techniques to graphics and To create 2D animations and 3D scenes
1908402.4	To create VR architecture using mobile and WebVR/XR
1908402.5	To create simple VR Game

CO- PO MATRIX

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
1908402.1	3	-	2	2	-	-	-	-	-	-	-	-
1908402.2	2	-	3	3	2	-	-	-	-	-	-	-
1908402.3	-	3	3	3	3	-	-	-	-	-	-	-
1908402.4	-	3	3	3	3	-	-	-	-	-	-	-
1908402.5	3	3	3	3	3	-	-	-	-	-	-	-
Average	3	3	3	3	3	-	-	-	-	-	-	-

CO- PSO MATRIX

1908305	PSO 1	PSO 2
1908402.1	-	2
1908402.2	-	2
1908402.3	-	3
1908402.4	-	2
1908402.5	-	3
Average	-	3

EVALUATION PROCEDURE FOR EACH EXPERIMENT

S.No	Description	Mark
1.	Aim & Pre-Lab discussion	20
2.	Observation	20
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	20
	Total	**100**

INTERNAL ASSESSMENT FOR LABORATORY

S.No	Description	Mark
1.	Conduction & Execution of Experiment	30
2.	Record	10
3.	Model Test	20
	Total	**50**

Ex.No:1

BASIC GRAPHICS FUNCTIONS

Aim:

To study Basic Graphics Functions

Graphics Functions:

1. initgraph()

initgraph() function initializes the graphics mode and clears the screen.

Declaration:

void far initgraph(int far *driver, int far *mode, char far *path)

2. detectgraph()

Detectgraph function determines the graphics hardware in the system, if the function finds a graphics adapter then it returns the highest graphics mode that the adapter supports.

Declaration:

void far detectgraph(int far *driver, int far *mode)

Integer that specifies the graphics driver to be used. You can give graphdriver a value using a constant of the graphics_drivers enumeration type.

3. closegraph()

closegraph() function switches back the screen from graphcs mode to text mode. It clears the screen also.

A graphics program should have a closegraph function at the end of graphics. Otherwise DOS screen will not go to text mode after running the program.

4. getpixel()

getpixel function returns the color of pixel present at location(x, y).

Declaration :-

int getpixel(int x, int y);

5. putpixel()

putpixel function plots a pixel at location (x, y) of specified color.

Declaration :-

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

For example if we want to draw a GREEN color pixel at (35, 45) then we will write putpixel(35, 35, GREEN); in our c program, putpixel function can be used to draw circles, lines and ellipses using various algorithms.

6. line()

line function is used to draw a line from a point(x1,y1) to point(x2,y2) i.e. (x1,y1) and (x2,y2) are end points of the line.

Declaration :-

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

7. lineto()

lineto function draws a line from current position(CP) to the point(x,y), you can get current position using getx and gety function.

8. circle()

circle function is used to draw a circle with center (x,y) and third parameter specifies the radius of the circle.

Declaration :-

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

9. ellipse()

Ellipse is used to draw an ellipse (x,y) are coordinates of center of the ellipse, stangle is the starting angle, end angle is the ending angle, and fifth and sixth parameters specifies the X and Y radius of the ellipse. To draw a complete ellipse strangles and end angle should be 0 and 360 respectively.

Declaration :-

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

10. drawpoly()

drawpoly function is used to draw polygons i.e. triangle, rectangle, pentagon, hexagon etc.

Declaration :-

void drawpoly(int num, int *polypoints);

num indicates (n+1) number of points where n is the number of vertices in a polygon, polypoints points to a sequence of (n*2) integers . Each pair of integers gives x and y coordinates of a point on the polygon. We specify (n+1) points as first point coordinates should be equal to (n+1)th to draw a complete figure.

To understand more clearly we will draw a triangle using drawpoly, consider for example the array :-

int points[] = { 320, 150, 420, 300, 250, 300, 320, 150};

points array contains coordinates of triangle which are (320, 150), (420, 300) and (250, 300). Note that last point(320, 150) in array is same as first.

11. outtext ()

outtext function displays text at current position.

Declaration :-

```
void outtext(char *string);
```

12. outtextxy ()

outtextxy function display text or string at a specified point(x,y) on the screen.

Declaration :-

```
void outtextxy(int x, int y, char *string);
```

x, y are coordinates of the point and third argument contains the address of string to be displayed.

13. rectangle()

Rectangle function is used to draw a rectangle. Coordinates of left top and right bottom corner are required to draw the rectangle. left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner.

Declaration :-

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

14. floodfill()

floodfill function is used to fill an enclosed area. Current fill pattern and fill color is used to fill the area.(x, y) is any point on the screen if (x,y) lies inside the area then inside will be filled otherwise outside will be filled,border specifies the color of boundary of area.

Declaration :-

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

15. fillpoly()

f illpoly function draws and fills a polygon. It require same arguments as drawpoly.

Declaration :-

```
void drawpoly( int num, int *polypoints );
```

16. fillellipse()

f illellipse function draws and fills a polygon.

Declaration:-

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

x and y are coordinates of center of the ellipse, xradius and yradius are x and y radius of ellipse respectively.

RESULT

Thus the study of basic graphics functions has been completed successfully.

Ex.No.2

BASIC OUTPUT PRIMITIVES

Aim:

To write a C Program to display the output primitives.

PRE LAB DISCUSSION

Graphics Primitive is a basic object that is essential for the creation or construction of complex images. Graphics is constructed from three basic elements, as opposed to the great variety of graphics applications. The most basic of these elemental structures is the pixel, short for picture element.

Several primitive elements can be combined together to create a complex image. Graphics primitives include: Lines, Circles, Arcs, Rectangles, etc.

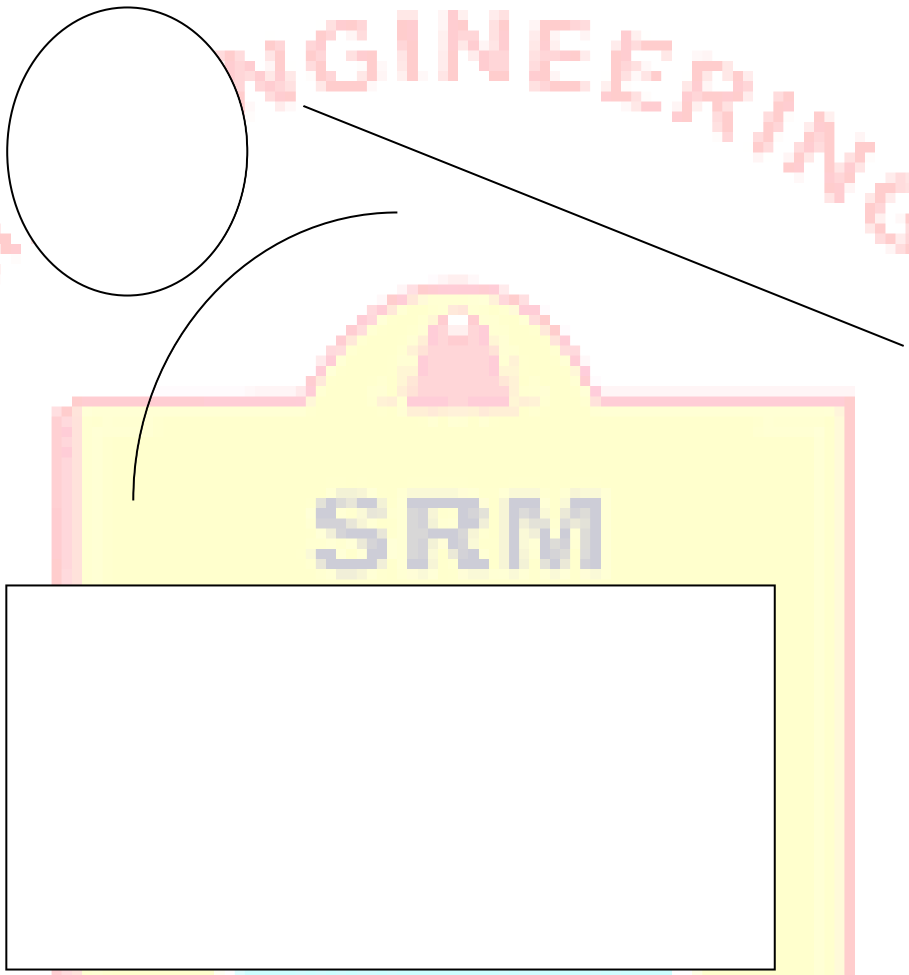
Algorithm:

```
1. Start the program .
2. Initialize the variables.
3. Call the initgraph() function
4. Set color for the output primitives.
5. Using Outtextxy() display the choosen particular primitives.
6. Using switch case mention the various primitives and their attributes.
7. The various primitives are arc, line ,circle, rectangle and ellipse.
8. close the graph and run the program.
9. stop the program.
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<string.h>
void main()
{
char ch='y';
int gd=DETECT,gm,x1,y1,x2,y2,rad,sa,ea,xrad,yrad,i;
initgraph(&gd,&gm,"");
while(ch=='y')
{
cleardevice();
setbkcolor(9);
outtextxy(100,150,"Enter 1 to get line");
outtextxy(100,170,"2.Circle");
outtextxy(100,190,"3.Box");
outtextxy(100,210,"4.Arc");
outtextxy(100,230,"5.Ellipse");
outtextxy(100,250,"6.Rectangle");
outtextxy(100,270,"7.Exit");
ch=getch();
cleardevice();
switch(ch)
{
case '1':
line(100,200,300,400);
break;
case '2':
circle(200,200,100);
break;
case '3':
setfillstyle(5,4);
bar(100,300,200,100);
break;
case '4':
setfillstyle(5,4);
arc(200,200,100,300,100);
break;
case '5':
setfillstyle(5,4);
fillellipse(100,100,50,100);
break;
case '6':
settextstyle(DEFAULT_FONT,0,2);
outtextxy(120,140,"VEL TECH");
line(100,100,100,300);
line(300,300,100,300);
line(100,100,300,100);
line(300,100,300,300);
break;
case '7':
closegraph();
return;
}
ch='y';
getch();
}
}
```

Output:



Result:

Thus the C program is executed successfully to display basic graphic primitives.

Ex.No:3a

IMPLEMENTATION OF ALGORITHMS FOR DRAWING 2D PRIMITIVES

DIGITAL DIFFERENTIAL ANALYZER

Aim:

To write a C Program for Line Drawing using Digital Differential Analyzer (DDA) Method.

PRE LAB DISCUSSION

DDA (Digital Differential Analyzer) is a line drawing algorithm used in computer graphics to generate a line segment between two specified endpoints. It is a simple and efficient algorithm that works by using the incremental difference between the x-coordinates and y-coordinates of the two endpoints to plot the line.

Algorithm:

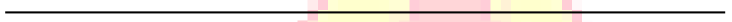
```
1. Start
2. Read the two end points of a line (x1, y1) & (x2, y2).
3. $\Delta x$ = x2-x1 & $\Delta y$ = y2-y1
4. Initialize (x, y) with (x1,y1)
        x= x1
        y= y1
5. if abs($\Delta$ x) > abs($\Delta$y) then
        steps =$\Delta$x
        else
        steps=$\Delta$y
6. xincrement =$\Delta$x / steps.
7. yincrement =$\Delta$y /steps
8. Plot the rounded coordinate (x, y)
9. Initialize counter k=1
10. Start the loop
      x = x + xincrement
      y = y+ yincrement
11. Plot the rounded coordinate(x, y)
12. Continue the loop till the counter k= steps
13. Stop.
```

PROGRAM:

```
#include "stdio.h"
#include "conio.h"
#include "math.h"
#include "graphics.h"
main()
{
        int gd=DETECT,gm;
        int xa,xb,ya,yb;
        int dx,dy,steps,k,xinc,yinc,x,y;
        initgraph(&gd,&gm,"c:\\tc\\bgi");
        printf("Enter the two left end pixel points(xa,ya):\n");
        scanf("%d%d",&xa,&ya);
        printf("Enter the two Right end pixel points(xb,yb):\n");
        scanf("%d%d",&xb,&yb);
        dx=xb-xa;
        dy=yb-ya;
        if(abs(dx)>abs(dy))
                steps=abs(dx);
        else
                steps=abs(dy);
        xinc=dx/steps;
        yinc=dy/steps;
        x=xa;
        y=ya;
        putpixel(x,y,6);
        for(k=1;k<=steps;k++)
        {
                x=x+xinc;
                y=y+yinc;
                putpixel(x,y,6);
        }
        getch();
        return(0);
  }
```

OUTPUT

```
Enter The Two Left Endpoints(xa,ya)  :     234     124

Enter The Two Right Endpoints(xb,yb) :     578     321
```



RESULT

Thus the C program to plot a line was executed successfully using DDA line drawing algorithm.

Ex.No:3b **BRESENHAMS LINE DRAWING**

Aim :

To write a C Program for Line Drawing using Bresenham's Line drawing Method.

PRELAB DISCUSSION

This algorithm helps us to perform scan conversion of a line. It is a powerful, useful, and accurate method. We use incremental integer calculations to draw a line. The integer calculations include addition, subtraction, and multiplication.

Algorithm:

```
1. Start
2. Read the two end points of a line (x1, y1) & (x2, y2).
3. Determine the Points which is used as start and end then store it in x and y
4. Plot the co-ordinate (x,y)
5. Calculate the constants Δx, Δy, 2Δy, 2Δy-2Δx and also obtain the starting value for the decision parameter as p=2Δy-Δx
6. At each xk along the line, starting at k=0, Perform the following :

    a. if pk<0, the next point to plot is (xk+1,yk) and pk+1=pk+2Δy
       otherwise
    b. the next point to plot is (xk+1,yk+1) and pk+1=pk+2Δy-2Δx
7. Repeat the Step 6 for Δx times
8. Stop.
```

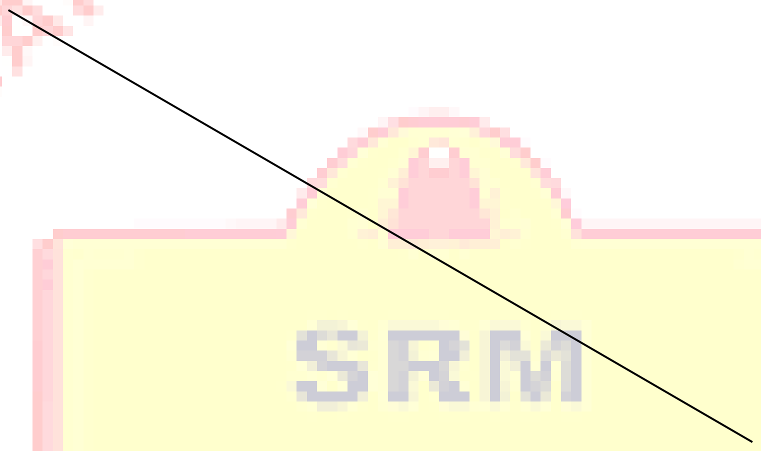
PROGRAM

```
#include "stdio.h"
#include "conio.h"
#include "math.h"
#include "graphics.h"
main()
{
        int gd=DETECT,gm;
        int xa,xb,ya,yb;
        int dx,dy,x,y,xend,p;
        initgraph(&gd,&gm,"c:\\tc\\bgi");
        printf("Enter The Two Left Endpoints(xa,ya):\n");
        scanf("%d%d",&xa,&ya);
```

```
            printf("Enter The Two Right Endpoints(xb,yb):\n");
            scanf("%d%d",&xb,&yb);
            dx=abs(xa-xb);
            dy=abs(ya-yb);
            p=2*dy-dx;
            if(xa>xb)
            {
                    x=xb;
                    y=yb;
                    xend=xa;
            }
            else
            {

                    x=xa;
                    y=ya;
                    xend=xb;
            }
        putpixel(x,y,6);

      while(x<xend)
        {
                x=x+1;
                if(p<0)
                {
                        p=p+2*dy;
                }
                else
                {
                        y=y+1;
                        p=p+2*(dy-dx);
                }
        putpixel(x,y,6);
        }
     getch();
     return(0);
   }
```

OUTPUT

```
Enter The Two Left Endpoints(xa,ya):        234    124

Enter The Two Right Endpoints(xb,yb):       578    321
```



RESULT

Thus the C Program for Line Drawing using Bresenham's Line drawing Method was executed and the output was verified successfully

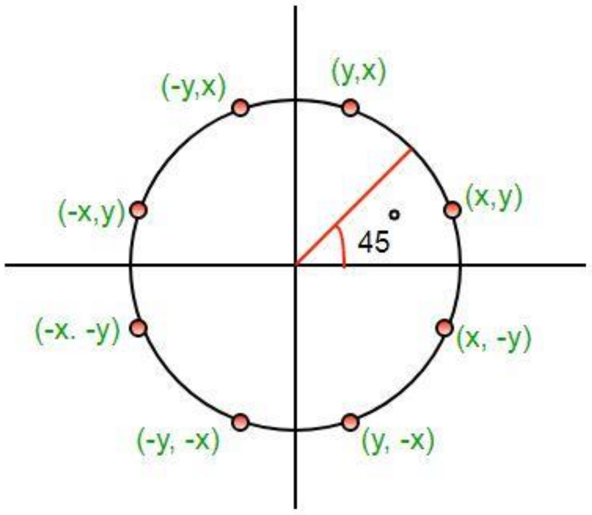
Ex.No:4 **IMPLEMENTATION OF MIDPOINT CIRCLE GENERATING ALGORITHM**

Aim :

To write a C Program for Circle Drawing using Midpoint Circle Generating algorithm.

PRE LAB DISCUSSION

The **mid-point** circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle. The **mid-point** algorithm to calculate all the perimeter points of the circle in the **first octant** and then print them along with their mirror points in the other octants. This will work because a circle is symmetric about its centre.



Algorithm:

```
1. Start
2. Get the center point (xcenter, ycenter) and radius(r) of a circle.
3. Initialize the variables
     i. x=0; y=r; p=1-r;
4. If (p<0) then
          i. p=p+(2*x)+1
else
     i. y--;
     ii. p=p+2* (x-y)+1
5. Repeat step 4 until x<y and Increment x by 1 each time.
6. Plot the pixel to display the circle.
7. Display the circle
8. Stop.
```

PROGRAM

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

```
#include<math.h>
int x,y,r,p,xcenter, ycenter;
void circleplot(int,int,int,int);
void main()
{
int gd = DETECT, gm;
initgraph(&gd,&gm,"..\\BGI:");
printf("\nEnter the x-coordinate of the centre  point :");
scanf("%d",&xcenter);
printf("\nEnter the y-coordinate of the centre point :");
scanf("%d",&ycenter);
printf("\nEnter the radius :");
scanf("%d",&r);
x=0;
y=r;
p=1-r;
while (x<y)
{
x++;
if (p<0)
p=p+2*x+1;
else
{
y--;
p=p+2*(x-y)+1;
}
circleplot(xcenter,ycenter,x,y);
}
getch();
closegraph();
}
void circleplot(int xcenter, int ycenter,int x, int y)
{
putpixel(xcenter+x,ycenter+y,10);
putpixel(xcenter-x,ycenter+y,10);
putpixel(xcenter+x,ycenter-y,10);
putpixel(xcenter-x,ycenter-y,10);
putpixel(xcenter+y,ycenter+x,10);
putpixel(xcenter-y,ycenter+x,10);
putpixel(xcenter+y,ycenter-x,10);
putpixel(xcenter-y,ycenter-x,10);
}
```

OUTPUT:

```
Enter the x-coordinate of the centre  point : 100
Enter the y-coordinate of the centre point : 100
Enter the radius : 50
```

RESULT

Thus C Program for Circle Drawing using Midpoint Circle Generating algorithm was executed successfully and the circle was plotted.

Ex.No. 5 2D GEOMETRIC TRANSFORMATIONS

Aim:

To write a C program to implement 2D Transformation – Translation, Rotation, Scaling, Shear and Reflection

PRE LAB DISCUSSION

1. Geometric Transformation: The object itself is transformed relative to the coordinate system or background. The mathematical statement of this viewpoint is defined by geometric transformations applied to each point of the object.
2. Coordinate Transformation: The object is held stationary while the coordinate system is transformed relative to the object. This effect is attained through the application of coordinate transformations.

An example that helps to distinguish these two viewpoints:

The movement of an automobile against a scenic background we can simulate this by

- Moving the automobile while keeping the background fixed-(Geometric Transformation)
- We can keep the car fixed while moving the background scenery- (Coordinate Transformation)

Algorithm:

```
1. Start
2. Get the coordinates of triangle (x1, y1, x2, y2, x3, y3).
3. Draw the original triangle.
4. Print the menu for choosing 2D Geometric Transformation.
    a. If users choose translation then get the translation factors(x, y) and draw the translated triangle
       in the following coordinates (x1+x, y1+y, x2+x, y2+y, x3+x, y3+y).
    b. If user choose rotation then
       Get the rotation angle (t) and reference point of the rotation (rx, ry).
        ▪ Change the t value to t = t * (3.14 / 180) and calculate the rotated coordinates by the
          following formulae rx1 = rx + (x1 - rx) * cos (t) - (y1 - ry) * sin (t); ry1 = ry + (x1 -
          rx) * sin (t) + (y1 - ry) *cos (t);
        ▪ Similarly calculate the coordinates rx2, ry2, rx3, ry3 and draw the rotated triangle in
          the following coordinates (rx1, ry1, rx2, ry2, rx3, ry3).
    c. If user choose scaling then
        • Get the scaling factors(x, y) and draw the scalded triangle in the following coordinates
          (x1*x, y1*y, x2*x, y2*y, x3*x, y3*y).
    d. If user choose reflection then
        • rotate the triangle in 1800 at (x2, y2) and draw the rotated triangle (which is reflected
          triangle).
    e. If user choose shearing then
        • Get the shear value and draw the sheared triangle in the following coordinates (x1,
          y1, x2+x, y2, x3, y3).
5. Stop
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
#include<math.h>
#include<stdlib.h>
void menu();
void input();
void output();
void translation();
void rotation();
void scaling();
void shearing();
void reflection();
int a[10][2],i,x,option,temp,angle,tx,ty,fx,fy,sh,k,n,axis,y;
float sx,sy;

void menu()
{
        printf("menu\n");
        printf("1.Translation\n");
        printf("2.rotation\n");
        printf("3.scaling\n");
        printf("4.shearing\n");
        printf("5.reflection\n");
        printf("6.exit\n");
        printf("enter the choice:");
        scanf("%d",&option);
        switch(option)
        {
                case  1:
                        input();
                        translation();
                        break;
                case 2:
                        input();
                        rotation();
                        break;
                case 3:
                        input();
                        scaling();
                        break;

                case 4 :
                        input();
                        shearing();
                break;
            case 5:
                input();
                reflection();
                break;
            case 6:
                exit(0);
                break;
        }
}

void input()
{
        printf("enter the number of vertices:" );
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
            printf("enter the coordinates:");
            scanf("%d%d%d%d",&a[i][0],&a[i][1],&a[i+1][0],&a[i+1][1]);
        }
}


void output()
{
        cleardevice();
        for(i=0;i<n;i++)
        {
            line(a[i][0],a[i][1],a[i+1][0],a[i+1][1]);
        }
}

void translation()
{
        output();
        printf("enter the tranformation vertex tx,ty:\n");
        scanf("%d%d",&tx,&ty);
        for(i=0;i<=n;i++)
        {
            a[i][0]=a[i][0]+tx;
            a[i][1]=a[i][1]+ty;
        }
        output();
        delay(10);
        menu();
}
```

```
void rotation()
{
        output();
        printf("enter the rotating angle:");
        scanf("%d",&y);
        printf("enter the pivot point:");
        scanf("%d%d",&fx,&fy);
        k=(y*3.14)/180;
        for(i=0;i<=n;i++)
        {
                a[i][0]=fx+(a[i][0]-fx)*cos(k)-(a[i][1]-fy)*sin(k);
                a[i][1]=fy+(a[i][0]-fx)*sin(k)-(a[i][1]-fy)*cos(k);
        }
        output();
        delay(10);
        menu();
}

void scaling()
{
        output();
        printf("enter the scaling factor\n");
        scanf("%f%f",&sx,&sy);
        printf("enter the fixed point:");
        scanf("%d%d",&fx,&fy);
        for(i=0;i<=n;i++)
        {
                a[i][0]=a[i][0]*sx+fy*(1-sx);
                a[i][1]=a[i][1]*sy+fy*(1-sy);
        }
        output();
        delay(10);
        menu();
}

void shearing()
{
        output();
        printf("enter the shear value:");
        scanf("%d",&sh);
        printf("enter the fixed point:");
        scanf("%d%d",&fx,&fy);
        printf("enter the axis for shearing if x-axis then 1 if y-axis the 0:");
        scanf("%d",&axis);
        for(i=0;i<=n;i++)
        {
                if(axis==1)
                {
                        a[i][0]=a[i][0]+sh*(a[i][1]-fy);
                }
                else
                {
                        a[i][1]=a[i][1]+sh*(a[i][0]-fx);
                }
        }
        output();
        delay(10);
        menu();
}

void reflection()
{
        output();
        for(i=0;i<=n;i++)
        {
                temp=a[i][0];
                a[i][0]=a[i][1];
                a[i][1]=temp;
        }
        output();
        delay(10);
        menu();
}


void main()
{
        int gd=DETECT,gm;
        initgraph(&gd,&gm,"c:\\tcplus\\bgi");
        menu();
        getch();
}
```

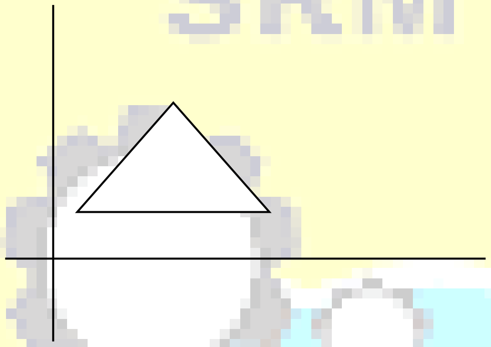
OUTPUT

Menu

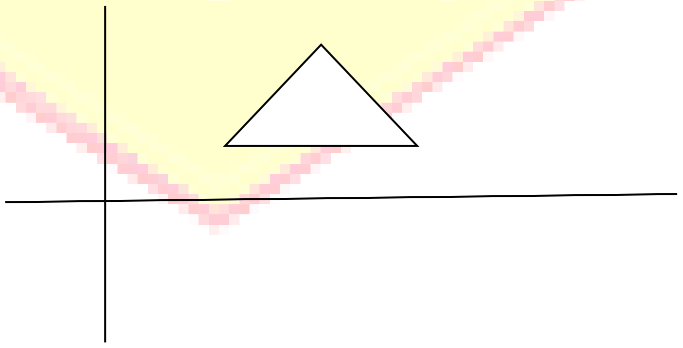
```
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection
6. Exit
```

TRANSLATION

```
Enter the choice : 1

Enter the number of Vertices: 3

Enter the coordinates :        30      150     10      200
Enter the coordinates :        10      200     60      200
Enter the coordinates :        60      200     30      150
```

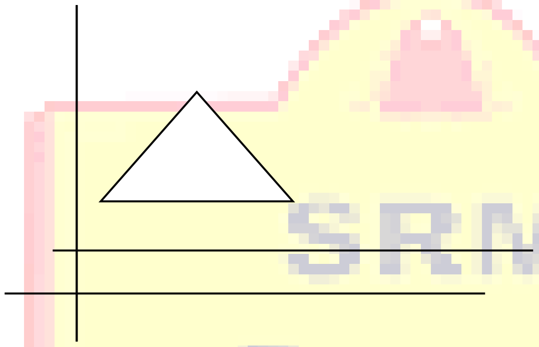


```
Enter the translation vector Tx, Ty : 90       60
```



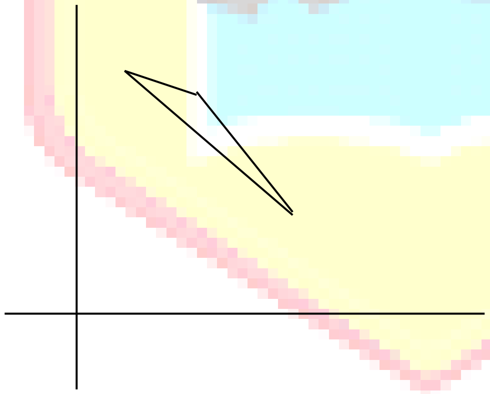
ROTATION

```
Enter the choice : 2

Enter the number of Vertices: 3

Enter the coordinates :        30      150     10      200
Enter the coordinates :        10      200     60      200
Enter the coordinates :        60      200     30      150
```



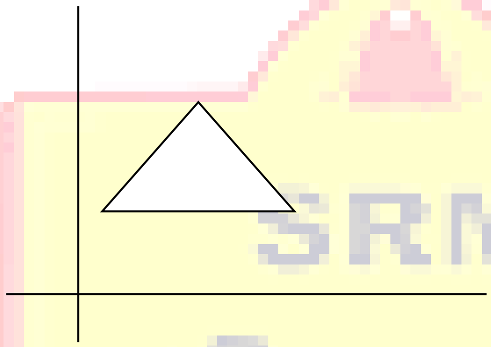
```
Enter the Rotating Angle :     90

Enter the Pivot Point    :     100     200
```

`

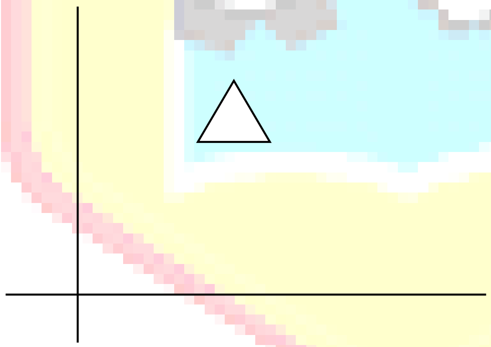


SCALING

```
Enter the choice : 3

Enter the number of Vertices: 3

Enter the coordinates :        30      150     10      200
Enter the coordinates :        10      200     60      200
Enter the coordinates :        60      200     30      150
```

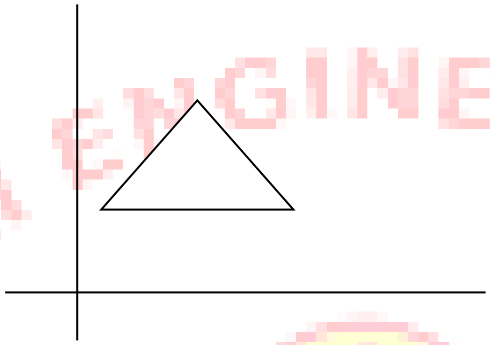


```
Enter the scaling Factor :     0.3     0.4
Enter the Fixed Point  :       100     200
```

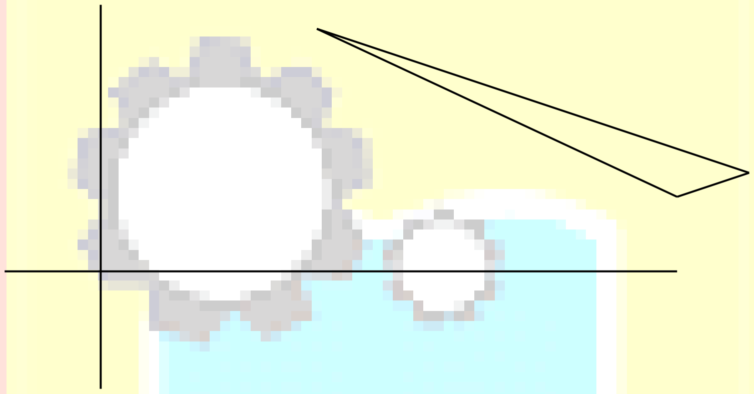


SHEARING

```
Enter the choice : 4

Enter the number of Vertices: 3
Enter the coordinates :          30      150     10      200
Enter the coordinates :          10      200     60      200
Enter the coordinates :          60      200     30      150
```

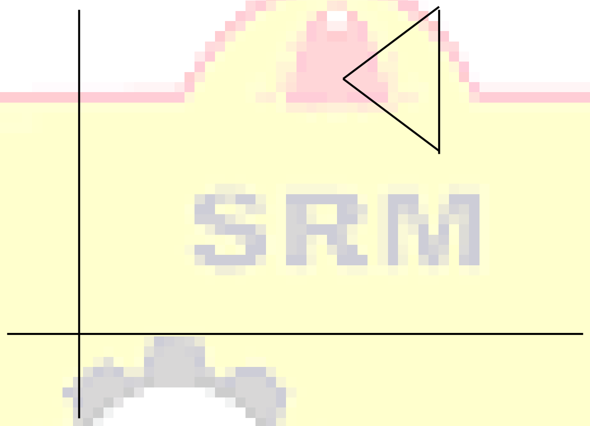


```
Enter the shear Value :          5
Enter the fixed point   : 50     100

Enter the Axis for shearing      if  x-axis  then  1
                                   if  y-axis  then  0
```



REFLECTION

```
Enter the choice : 5

Enter the number of Vertices: 3

Enter the coordinates :      30      150     10      200
Enter the coordinates :      10      200     60      200
Enter the coordinates :      60      200     30      150
```



RESULT

Thus C program to implement 2D Transformation of Translation, Rotation, Scaling, Shear and Reflection was executed and the output was verified successfully.

Ex.No.6 **IMPLEMENTATION OF LINE CLIPPING ALGORITHM**

Aim:

To study and Implement Cohen Sutherland 2D line clipping algorithm to clip the line against clip window.

PRE LAB DISCUSSION

A line-clipping algorithm processes each line in a scene through a series of tests and intersection calculations to determine whether the entire line or any part of it is to be saved. It also calculates the intersection position of a line with the window edges so its major goal is to minimize these calculations.

Line clipping is a clipping concept in which lines that lies outside the clipping window is removed from the clip region. As a result, only lines which is inside the view plane are visible. Cohen Sutherland Algorithm is one of the popular line clipping algorithm used for the purpose.

Algorithm:

1. Start
2. Input two endpoints of the line say p_1 (x_1 , y_1) and p_2 (x_2 , y_2)
3. Input two corners (Let-top and right -bottom) of the window , say(wx_1 ,wy_1 and wx_2 , wy_2)
4. Assign the region codes for two endpoints p_1 and p_2 using following steps :

Initialize code with bits 0000

Set Bit1 =0 if ($x < wx_1$)

Set Bit2 =0 if ($x < wx_2$)

Set Bit3 =0 if ($y < wy_2$)

Set Bit4 =0 if ($y < wy_1$)

5. Check for visibility of line

a. If region codes for both endpoints p_1 and p_2 are zero then the line is completely visible. Hence draw the line and go to step 9

b. If region codes for both endpoints are not zero and the logical ANDing of them is also nonzero then the line is completely invisible. So reject the line and go to 9

c. If region codes for two endpoints do not satisfy the condition in (4a and 4b the line is partially visible.

d. Determine the intersecting edge of the clipping window by inspecting the region codes of two endpoints

e. If region codes for both endpoints are non- zero, find intersecting point p'_1 and p'_2 with boundary edges of clipping window with respect to point p_1 and point p_2 respectively

f. If region codes for any one endpoints are non-zero, find intersecting point p'_1 or p'_2 with boundary edges of clipping window with respect to it.

6. Divide the Line segments considering intersection points
7. Reject the line segments if any one endpoint of it appears outsides the clipping window
8. Draw the remaining line segments
9. Stop

PROGRAM:

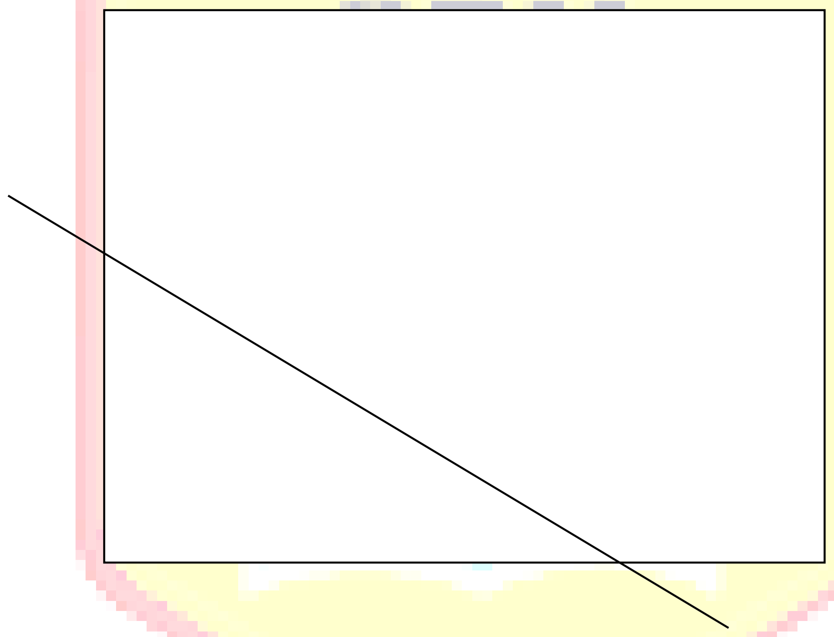
```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
float cxl,cxr,cyt,cyb;
code(float ,float);
void clip(float ,float,float,float);
void  rect(float ,float,float,float);
 main()
    {
                float x1,y1,x2,y2;
                int g=0,d;
                initgraph(&g,&d,"c:\\tc\\bin");
                settextstyle(1,0,1);
                outtextxy(40,15,"BEFORE CLIPPING");
                printf("\n Please Enter Left,Bottom,Right,Top Of Clip Window");
                scanf("%f%f%f%f",&cxl,&cyb,&cxr,&cyt);
rect(cxl,cyb,cxr,cyt);
                getch();
printf("\n Enter  The Line Coordinate");
                scanf("%f%f%f%f",&x1,&y1,&x2,&y2);
line(x1,y1,x2,y2);
                getch();
                cleardevice();
                settextstyle(1,0,1);
                outtextxy(40,15,"AFTER CLIPPING");
                clip(x1,y1,x2,y2);
                getch();
                closegraph();
   }

void clip(float x1,float y1,float x2,float y2)
 {
                int c,c1,c2;
                float x,y;
                c1=code(x1,y1);
                c2=code(x2,y2);
                getch();


                while((c1!=0)||(c2!=0))
                 {
                     if((c1&c2)!=0)
                     goto out;
```

```
                    c=c1;
                    if(c==0)
                            c=c2;
                    if((c&1)==1)
                      {
                            y=y1+(y2-y1)*(cxl-x1);
                            x=cxl;
                      }
                    else
                    if((c&2)==2)
                      {
                            y=y1+(y2-y1)*(cxl-x1)/(x2-x1);
                            x=cxr;
                      }
                    else
                    if((c&8)==8)
                      {
                            x=x1+(x2-x1)*(cyb-y1)/(y2-y1);
                            y=cyb;
                     }
                  else
                  if((c&4)==4)
                     {
                            x=x1+(x2-x1)*(cyt-y1)/(y2-y1);
                            y=cyt;
                     }
                  if(c==c1)
                    {
                            x1=x;
                            y1=y;
                            c1=code(x,y);
                    }
                  else
                    {
                            x2=x;
                            y2=y;
                            c2=code(x,y);
                    }
            }


out:
                    rect(cxl,cyb,cxr,cyt);
                    line(x1,y1,x2,y2);
   }
```

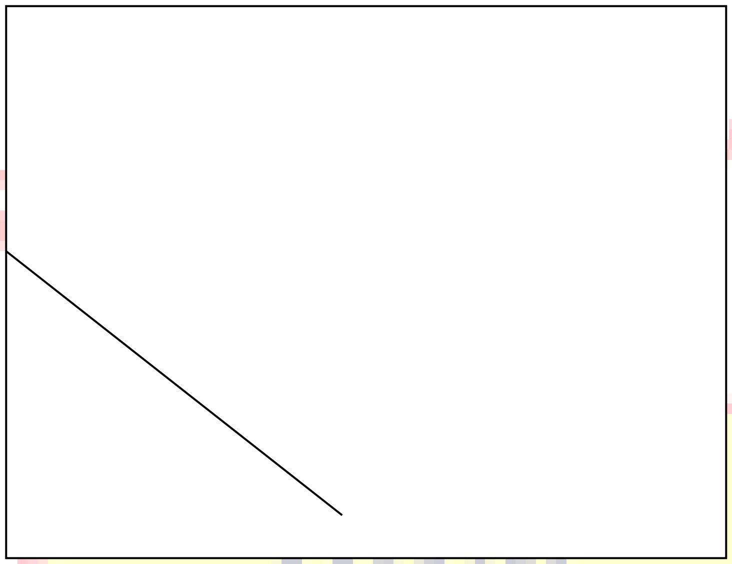
```
code(float x ,float y)
 {
                    int c=0;
                    if(x<cxl)                          c=1;
                    else
                              if(x>cxr)                 c=2;
                    else
                              if(y<cyb)                 c=c|8;
                    else
                              if(y>cyt)                 c=c|4;
                    return c;
}


void rect(float xl,float yb,float xr,float yt)
  {
                    line(xl,yb,xr,yb);
                    line(xr,yb,xr,yt);
                    line(xr,yt,xl,yt);
                    line(xl,yt,xl,yb);
  }
```

OUTPUT

```
BEFORE CLIPPING

Please Enter Left , Bottom , Right , Top Of The Clip window

200
200
400
400

Please Enter The Line Coordinates (X1, Y1, X2, Y2)

150
300
400
450
```



AFTER CLIPPING



RESULT:

Thus the C program to Implement Cohen Sutherland 2D line clipping algorithm to clip the line against clip window was executed and the output was verified successfully.

Ex.No. 7 3D TRANSFORMATIONS - TRANSLATION, ROTATION, SCALING

Aim :

To perform 3D transformations such as Translation, Rotation and Scaling on 3D object.

PRE LAB DISCUSSION

The geometric transformations play a vital role in generating images of three Dimensional objects with the help of these transformations. The location of objects relative to others can be easily expressed. Sometimes viewpoint changes rapidly, or sometimes objects move in relation to each other. For this number of transformation can be carried out repeatedly.

Algorithm :

```
Step 1: 3 dimensional transformation it has three axis x,y,z.
Step 2: Depending upon the coordinate it will perform the Translation,
        Rotation, Scaling,
Step 3: The translation can be performed by changing the sign of the translation
        components Tx, Ty, and Tz.
Step 4: Perform the scaling of 3D object with scaling parameter
Step 5: Increase of rotation, object can be rotated about x or y or z axis.
Step 6: Display the transmitted object in the screen
```

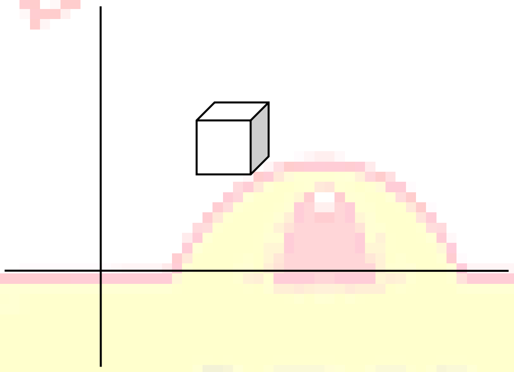
PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
int maxx,maxy,midx,midy;
void axis()
{
        getch();
        cleardevice();
        line(midx,0,midx,maxy);
        line(0,midy,maxx,midy);
}

void main()
{
        int gd,gm,x,y,z,o,x1,x2,y1,y2;
        detectgraph(&gd,&gm);
        initgraph(&gd,&gm," ");
        setfillstyle(0,getmaxcolor());
        maxx=getmaxx();
        maxy=getmaxy();
        midx=maxx/2;
        midy=maxy/2;
        axis();
        bar3d(midx+50,midy-100,midx+60,midy-90,5,1);
        printf("Enter Translation Factor");
        scanf("%d%d%d",&x,&y,&z);
        axis();
        printf("after translation");
        bar3d(midx+(x+50),midy-(y+100),midx+x+60,midy-(y+90),5,1);
        axis();
        bar3d(midx+50,midy+100,midx+60,midy-90,5,1);
        printf("Enter Scaling Factor");
        scanf("%d%d%d",&x,&y,&z);
        axis();
        printf("After Scaling");
        bar3d(midx+(x*50),midy-(y*100),midx+(x*60),midy-(y*90),5*z,1);
        axis();
        bar3d(midx+50,midy-100,midx+60,midy-90,5,1);
        printf("Enter Rotating Angle");
        scanf("%d",&o);
        x1=50*cos(o*3.14/180)-100*sin(o*3.14/180);
        y1=50*cos(o*3.14/180)+100*sin(o*3.14/180);
        x2=60*sin(o*3.14/180)-90*cos(o*3.14/180);
        y2=60*sin(o*3.14/180)+90*cos(o*3.14/180);
        axis();
        printf("After Rotation about Z Axis");
        bar3d(midx+x1,midy-y1,midx+x2,midy-y2,5,1);
        axis();
        printf("After Rotation about X Axis");
        bar3d(midx+50,midy-x1,midx+60,midy-x2,5,1);
        axis();
        printf("After Rotation about Y Axis");
        bar3d(midx+x1,midy-100,midx+x2,midy-90,5,1);
        getch();
        closegraph();
}
```

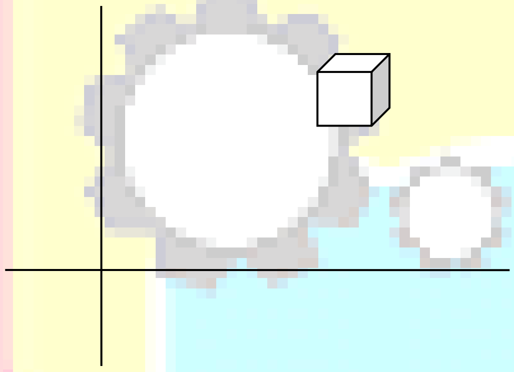
OUTPUT

Translation

Enter Translation Factor : 50 60 70

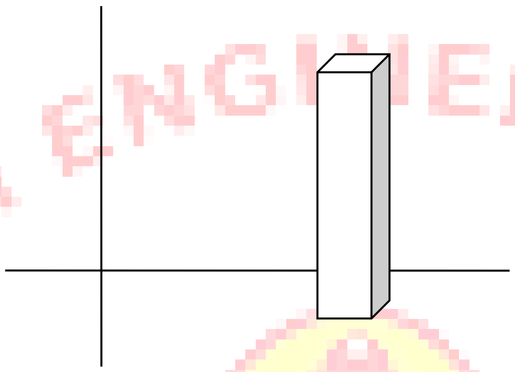


After Translation

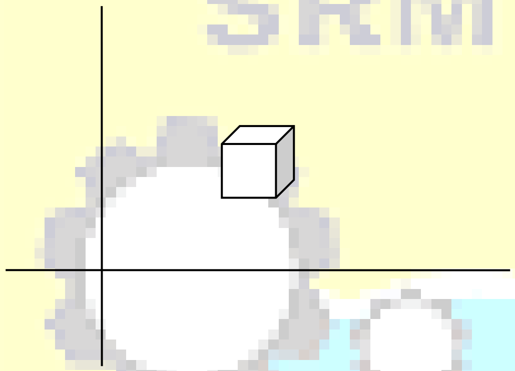


Scaling

```
Enter Scaling Factor :  80 90 95
```

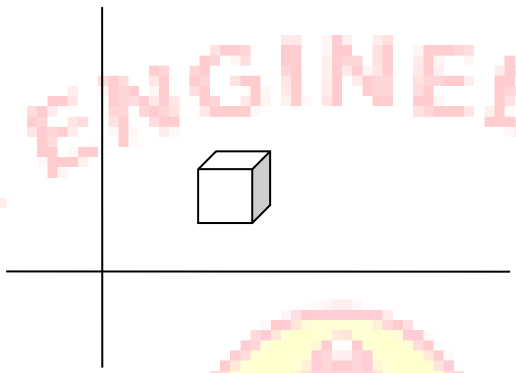


After Scaling

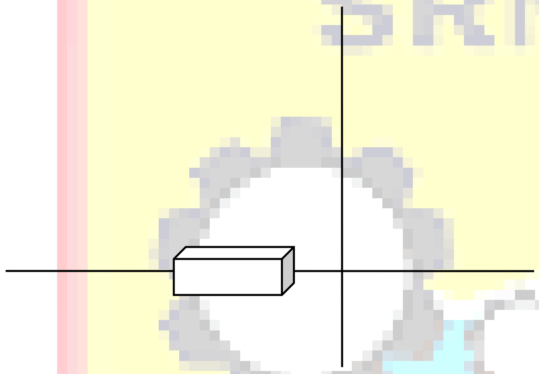


Rotation

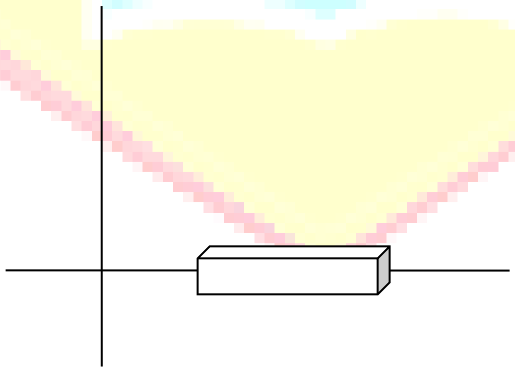
```
Enter Rotating Angle : 60
```



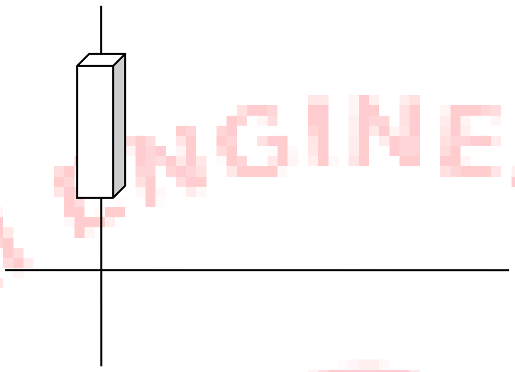
After Rotation about Z-Axis



After Rotation about X-Axis



After Rotation about Y-Axis :



RESULT :

Thus the C program to perform 3D transformations such as Translation, Rotation and Scaling on 3D object was executed and output was verified successfully.

EX NO : 8

CREATING 3 DIMENSIONAL SCENE

AIM

To create 3 dimensional scene using blender.

PRE LAB DISCUSSION

Blender is a powerful open-source 3D modeling software used in various industries, including animation, game development, architecture, and more.

- **Interface**: Understand Blender's interface and navigation tools.
- **Modeling Process:** Focus on polygon modeling for this lab.
- **Reference Images**: Gather images or sketches to guide your design.
- **Basic Mesh Editing:** Learn essential mesh manipulation tools (extrusion, scaling, etc.).
- **Subdivision Surfaces**: Use subdivision to smooth your model.
- **Modifier Stack:** Apply non-destructive modifications to your model.

Procedure

Step 1: Reference Images

Gather reference images or sketches of the sofa you want to create.

Step 2: Setting Up Blender

Open Blender and delete the default cube by selecting it and pressing "Delete" on your keyboard. Download Blender kit from browser.

Step 3: Add a Cube

Press Shift + A, then select "Mesh" > "Cube" to add a cube to the scene. This cube will form the base of the sofa.

Step 4: Edit Mode

Select the cube and press Tab to enter Edit Mode. Now, you can manipulate its vertices, edges, and faces.

Step 5: Extrude and Shape the Sofa

Select Shift + Spacebar + S to extrude or Select Scale at your left of the blender module scale tools to start shaping the sofa. Extrude the plane upward to create the backrest and armrests.

Step 6: Add Cushions

Extrude and scale additional faces to create cushions on the seating area and backrest. Use loop cuts

(Ctrl + R) to add more geometry to achieve a rounded cushion effect.

Step 7: Use the Scale Modifier

With the sofa object selected, go to the "Modifiers" tab in the Properties panel (the wrench icon). Click on "Add Modifier" and choose "Subdivision Surface" from the list.

Step 8: Configure the Scale Modifier

At the top left Change the Object mode to Edit mode.**Step 9: Reshaping the cube**

Select Shift + Spacebar or Ctrl + R to reshape or resizing the cube based on your prefrences.

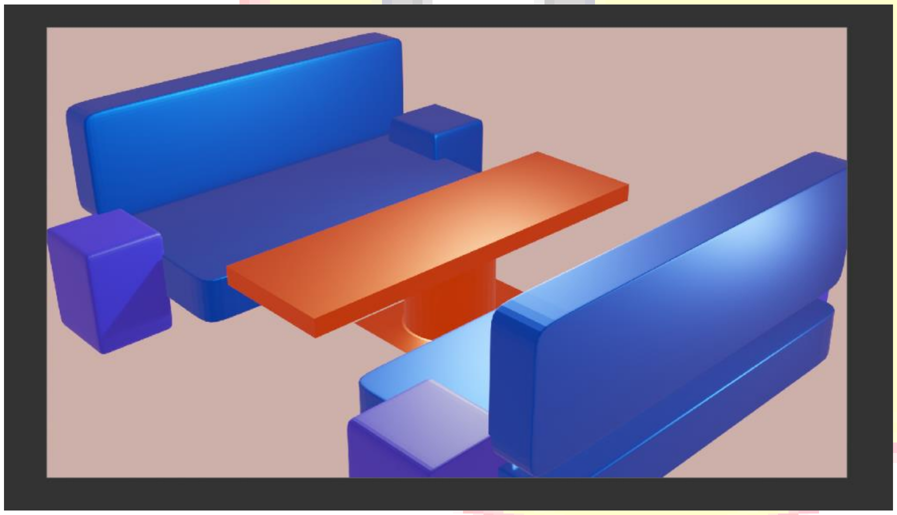
Step 10: Repeat the Process

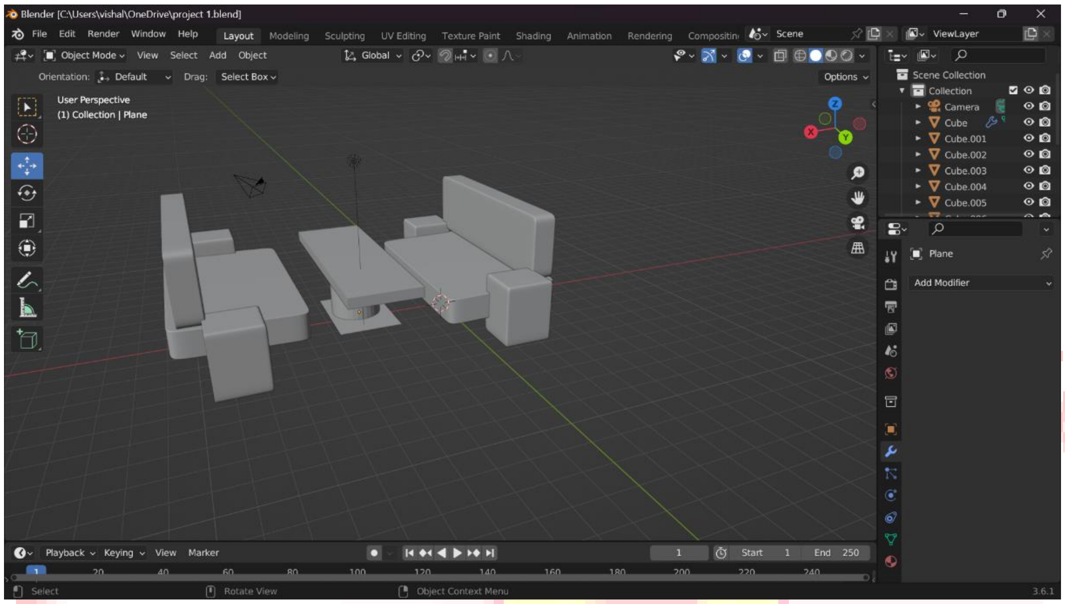
Repeat the process from Step 4 based on your desired image.

Step 11: Finalize and Render

Inspect your model and make any additional adjustments. Once you're satisfied, go to the "Render" tab to set up your scene's lighting, camera angle, and other settings. Press F12 or click "Render" to create the final rendered image of your sofa.

OUTPUT




Blender [C:\Users\vishal\OneDrive\project 1.blend]
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing
Scene
ViewLayer
Object Mode View Select Add Object
Global
Orientation: Default Drag: Select Box
Options
User Perspective
(1) Collection | Plane
Scene Collection
Collection
Camera
Cube
Cube.001
Cube.002
Cube.003
Cube.004
Cube.005
Plane
Add Modifier
Playback Keying View Marker
Start 1 End 250
Select
Rotate View
Object Context Menu
3.6.1

Result

Thus to create 3 dimensional model using blender has been executed .

EX NO : 09 2D ANIMATION – TO CREATE INTERACTIVE ANIMATION USING ANY AUTHORING TOOL

AIM

To create 2 dimensional animation using blender.

PRE LAB DISCUSSION

Blender is a powerful open-source 3D modeling software used in various industries, including animation, game development, architecture, and more.

- **Interface**: Understand Blender's interface and navigation tools.
- **Modeling Process:** Focus on polygon modeling for this lab.
- **Reference Images**: Gather images or sketches to guide your design.
- **Basic Mesh Editing:** Learn essential mesh manipulation tools (extrusion, scaling, etc.).
- **Subdivision Surfaces**: Use subdivision to smooth your model.
- **Modifier Stack:** Apply non-destructive modifications to your model.

Procedure

1. Modelling the Car:

a. Start a new Blender project.

b. Delete the default cube (`X` to delete).

c. Add a basic car shape using cubes, cylinders, and other basic shapes. (Alternatively, you can import a car model if you have one or download from free repositories like [Blend Swap](https://www.blendswap.com/)).

d. Make sure all parts of your car are combined into one object using `Ctrl + J`.

2. Setting Up the Scene:

a. Place the car model at the starting point of your animation.

b. Set up the camera angle. Select the camera, click `G` to move and `R` to rotate it. Aim it towards the path of the car.

3. Animating the Car:

a. Go to frame 1 in the timeline (usually at the bottom of the Blender window).

b. Select the car. click `I` and choose `Location`. This inserts a keyframe for the car's location at frame 1.

c. Move the timeline's green line (current frame indicator) to a later frame, say frame 100.

d. Move the car to the desired ending location.

e. Again, hit `I` and choose `Location`. This inserts another keyframe for the car's location at frame 100.

f. Play the animation using the Play button or pressing `Space`. You'll see the car move from its start to end position between frames 1 and 100.

4. Adding More Realism (optional steps):

a. Simple Rotation: If the car turns during its movement, you'll want to keyframe its rotation as well.

b. Using a Path: For more complicated paths, use a curve as a path for the car. Parent the car to the curve with a 'Follow Path' constraint. Adjust the 'Offset' to animate.

c. Wheel Rotation: For added realism, you can animate the wheels rotating. This can be done manually (rotating and keyframing) or using drivers to make the wheels rotate based on the car's movement.

5. Rendering:

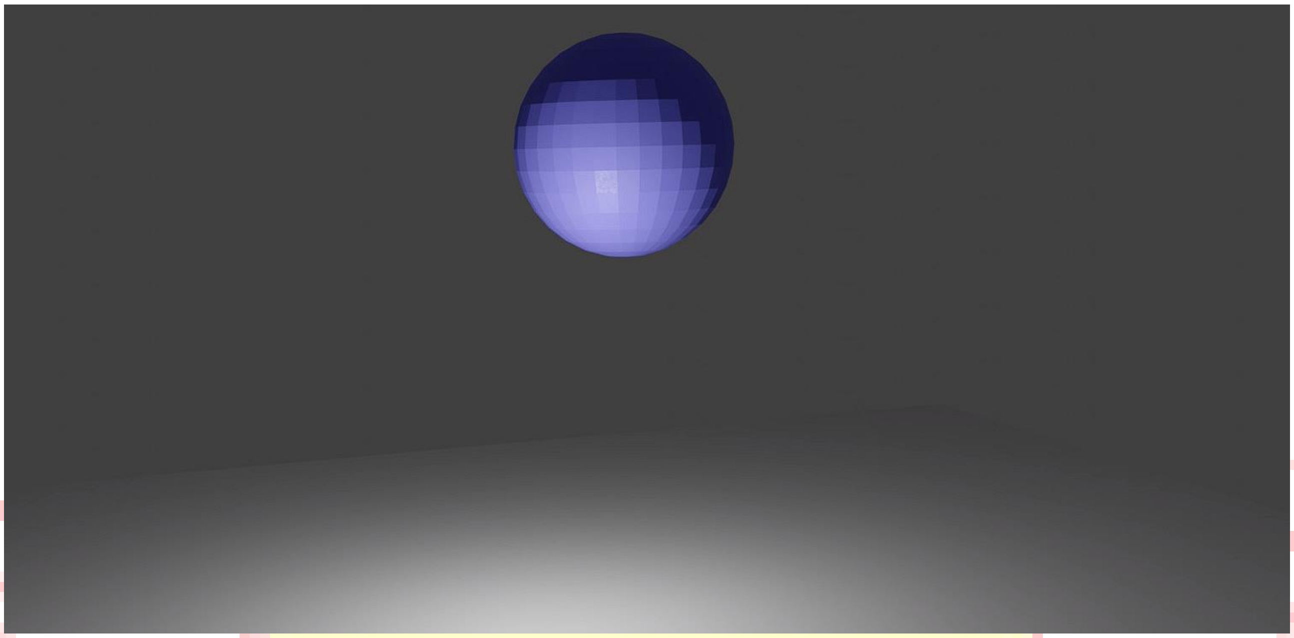
a. Go to the Render tab on the right side panel.

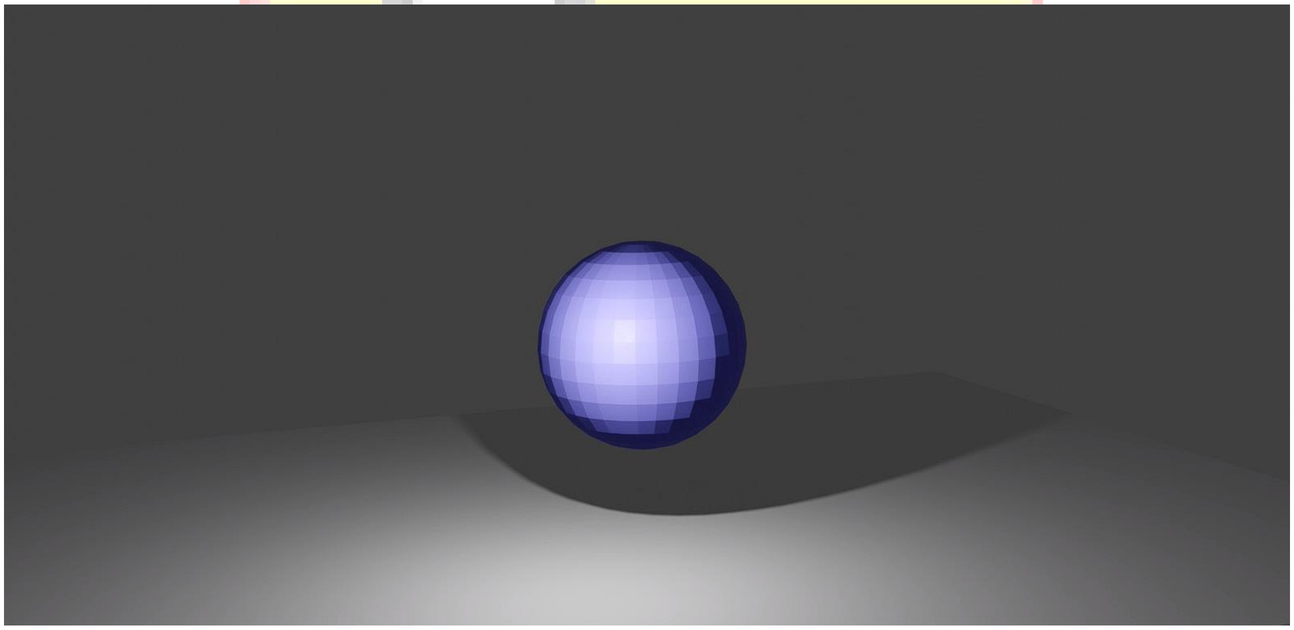
b. Choose your desired output settings (e.g., resolution, file format).

c. Click the `Render Animation` button.

Remember, this is a basic guideline. Blender is a powerful tool, and there are countless ways to achieve effects, from using physics simulations to more complex rigging techniques. To truly harness the power of Blender for animations, consider watching video tutorials or reading comprehensive guides on Blender animation.

OUTPUT





RESULT :

Thus the 2 dimensional animation using blender has been created successfully .

EX NO : 10 HAND HELD VR MOBILE APP DEVELOPMENT USING UNITY

Aim:

To develop Hand held VR mobile app using unity

Procedure:

Handheld VR mobile app development using unity

Set up Unity: Download and install the Unity game development platform from the official website (https://unity.com/). Ensure you have the necessary hardware and software requirements to run Unity.

Create a new project: Launch Unity and create a new project for your VR app. Choose a name and location for the project files.

Configure VR settings: Unity supports multiple VR platforms, such as Oculus, HTC Vive, and Google Cardboard. Set up the appropriate VR platform for your handheld VR app by installing the necessary Unity packages or SDKs provided by the platform.

Design your virtual environment: Create a virtual environment where users can interact in VR. This includes creating 3D models, importing assets, and designing the scene layout. Unity provides a visual editor where you can manipulate objects and position them in the scene.

Implement interactivity: Add interactivity to your VR app by scripting behaviors using C# or Unity's visual scripting system (e.g., Unity's Playmaker or Bolt). This involves defining how objects should respond to user inputs, such as gaze, touch, or controller interactions.

Integrate VR controls: Implement VR controls that allow users to navigate and interact with the virtual environment. This could involve gaze-based interactions, hand tracking, or using external controllers. Unity provides APIs and packages to facilitate VR input handling.

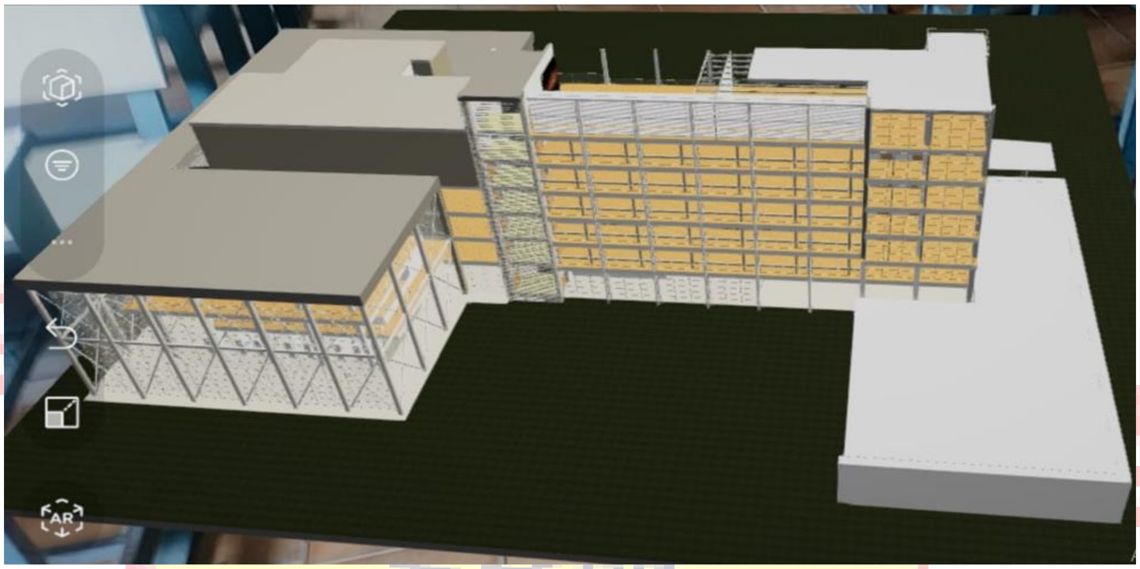
Optimize performance: VR applications require high performance to maintain a smooth and immersive experience. Optimize your app by minimizing the number of draw calls, using efficient shaders, and optimizing the use of resources.

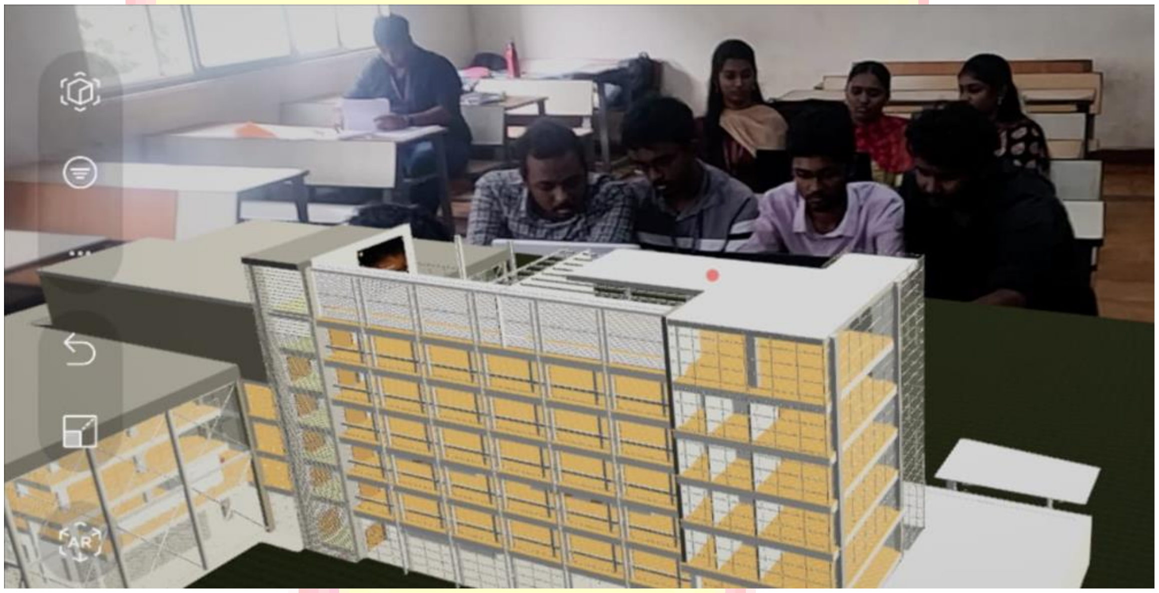
Test and iterate: Test your app on actual handheld VR devices to ensure it functions correctly. Identify and fix any bugs or performance issues. Iterate on your design based on user feedback to improve the user experience.

Build and deploy: Once you are satisfied with your app, build it for the target mobile platforms (iOS or Android). Unity provides options to build and deploy your app directly to your connected mobile device or generate installation files for distribution.

Publish and distribute: If you want to make your app available to the public, you can publish it on app stores such as the Apple App Store or Google Play Store. Follow the respective guidelines and requirements for each store to submit your app for review and distribution.

OUTPUT:





.

Result:

Thus Hand held VR mobile app using unity was developed

EX NO : 11 To create a VR walkthrough using Unity for mobile apps, web VR, or XR

Procedure:

1. Go to the Artsteps website and sign in with your Google account.
2. Click on the "Create New Project" button.
3. Enter a name for your project and click on the "Create" button.

Importing Your Space

Once you have created a new project, you can import your space into Artsteps. There are two ways to do this:

Creating the Walkthrough

Once you have imported your space into Artsteps, you can create the walkthrough. To do this, follow these steps:

1. Click on the "Walkthrough" tab.
2. Click on the "Add Point" button.
3. Click on a location in your space where you want to add a point.
4. Repeat steps 2 and 3 until you have added points to all of the locations in your space that you want to include in the walkthrough.
5. To connect the points, click on the "Connect Points" button.
6. Click on a point and then click on the next point to connect them.
7. Repeat step 6 until all of the points are connected.

Adding Hotspots and Information

Upload a 3D model: If you have already created a 3D model of your space, you can upload it to Artsteps. Artsteps supports a variety of 3D file formats, including OBJ, FBX, and GLTF.

Take a 360-degree photo: If you do not have a 3D model of your space, you can take a 360-degree photo and import it into Artsteps. Artsteps will automatically convert the photo into a 3D model.

You can add hotspots and information to your walkthrough to make it more informative and engaging. To do this, follow these steps:

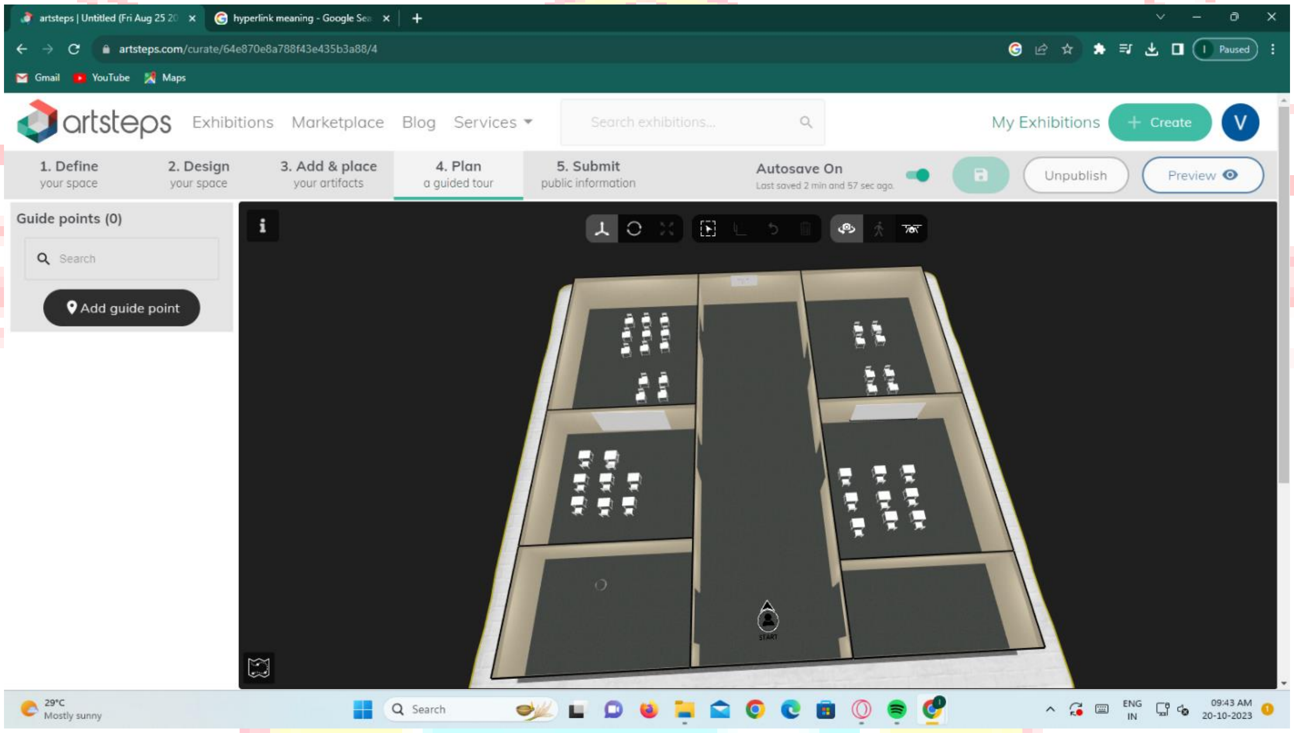
Add hotspots: Click on the "Add Hotspot" button. Then, click on a location in your space where you want to add a hotspot.

Add information: Click on the "Add Info" button. Then, enter the information that you want to add to the hotspot.

Previewing and Publishing the Walkthrough

Once you have finished creating the walkthrough, you can preview it to see how it will look. To do this, click on the "Preview" button.

Output:



Result:

Thus VR walkthrough has been generated using Unity for mobile apps.

EX NO : 12 **CREATING 3 DIMENSIONAL MODEL**

Aim

To create 3-dimensional model using blender.

Pre-lab Discussion

Blender is a powerful open-source 3D modeling software used in various industries, including animation, game development, architecture, and more.

- **Interface**: Understand Blender's interface and navigation tools.
- **Modeling Process:** Focus on polygon modeling for this lab.
- **Reference Images**: Gather images or sketches to guide your design.
- **Basic Mesh Editing:** Learn essential mesh manipulation tools (extrusion, scaling, etc.).
- **Subdivision Surfaces**: Use subdivision to smooth your model.
- **Modifier Stack:** Apply non-destructive modifications to your model.

Procedure :

Step 1: Reference Images

Gather reference images or sketches of the sofa you want to create.

Step 2: Setting Up Blender

Open Blender and delete the default cube by selecting it and pressing "Delete" on your keyboard.

Step 3: Add a Cube

Press Shift + A, then select "Mesh" > "Cube" to add a cube to the scene. This cube will form the base of the sofa.

Step 4: Edit Mode

Select the cube and press Tab to enter Edit Mode. Now, you can manipulate its vertices, edges, and faces.

Step 5: Extrude and Shape the Sofa

Select Shift + Spacebar + S to extrude or Select Scale at your left of the blender module scale tools to start shaping the sofa. Extrude the plane upward to create the backrest and armrests.

Step 6: Add Cushions

Extrude and scale additional faces to create cushions on the seating area and backrest. Use loop cuts (Ctrl + R) to add more geometry to achieve a rounded cushion effect.

Step 7: Use the Scale Modifier

With the sofa object selected, go to the "Modifiers" tab in the Properties panel (the wrench icon). Click on "Add Modifier" and choose "Subdivision Surface" from the list.

Step 8: Configure the Scale Modifier

At the top left Change the Object mode to Edit mode.

Step 9: Reshaping the cube

Select Shift + Spacebar or Ctrl + R to reshape or resizing the cube based on your prefrences.

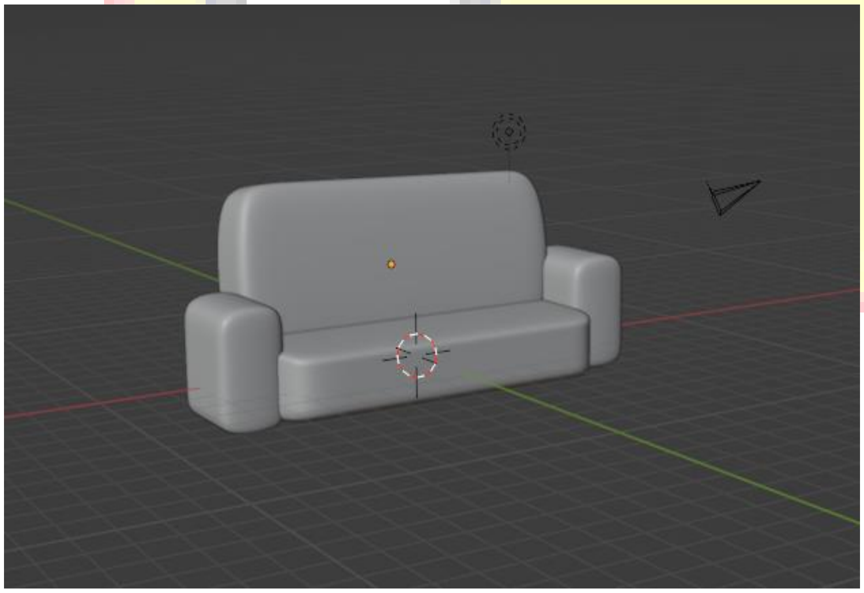
Step 10: Repeat the Process

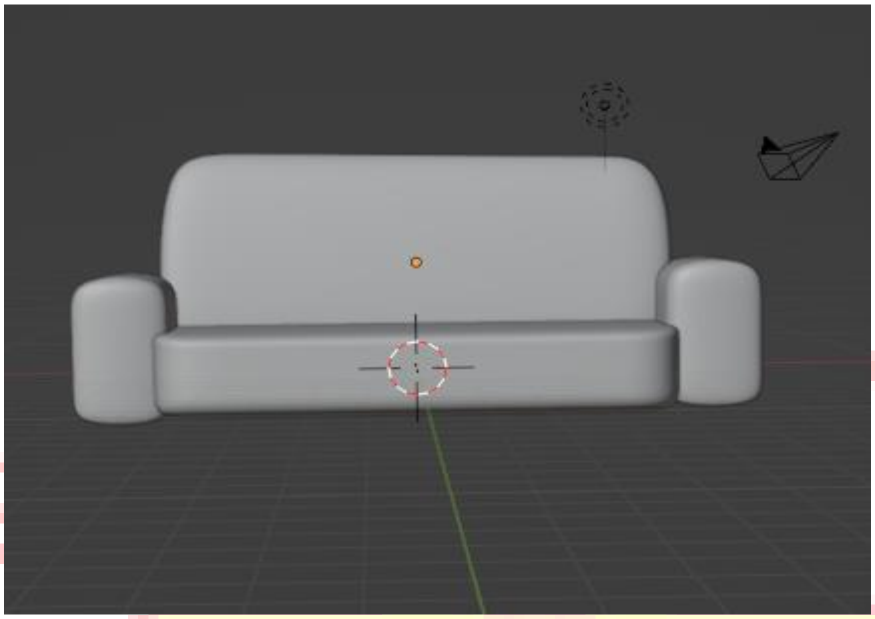
Repeat the process from Step 4 based on your desired image.

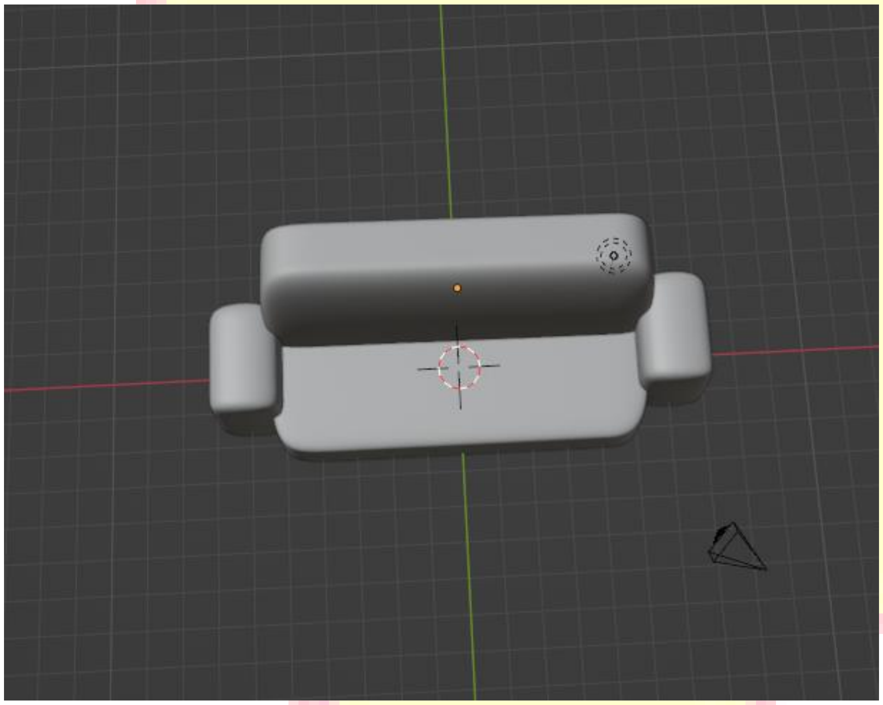
Step 11: Finalize and Render

Inspect your model and make any additional adjustments. Once you're satisfied, go to the "Render" tab to set up your scene's lighting, camera angle, and other settings. Press F12 or click "Render" to create the final rendered image of your sofa.

Output







Result:

Thus three dimensional model was created using blender.

EX NO : 13 **Developing a VR game using Unity**

Aim:

To Develop VR Game using Unity

Procedure:

Set up Unity: Download and install the Unity game development platform from the official website (https://unity.com/). Ensure you have the necessary hardware and software requirements to run Unity.

Plan your game: Define the concept, gameplay mechanics, and visual style of your VR game. Create a design document or a storyboard to outline the main features and elements of the game.

Create a new project: Launch Unity and create a new project for your VR game. Choose a name and location for the project files.

Set up VR platform: Determine the target VR platform for your game. Unity supports various platforms such as Oculus, HTC Vive, or PlayStation VR. Install the necessary Unity packages or SDKs provided by the platform to support VR functionality.

Design the game environment: Create the virtual environment where the game will take place. This includes designing the scene layout, importing 3D models, and adding appropriate lighting and textures. Unity provides a visual editor where you can manipulate objects and position them in the scene.

Implement gameplay mechanics: Add gameplay mechanics specific to your VR game. This can include interactions with objects, enemy AI, physics-based puzzles, or any other mechanics that fit your game concept. Use C# scripting or Unity's visual scripting tools to define the behavior of objects and characters in the game.

Implement VR interactions: Use Unity's VR input system or platform-specific APIs to implement VR interactions. This includes handling user input from VR controllers or hand tracking, detecting object interactions, and triggering actions based on user gestures or movements.

Audio design: Create and implement audio elements for your VR game, such as background music, sound effects, and spatial audio. Consider the immersive nature of VR and use audio cues to enhance the player's experience and provide feedback.

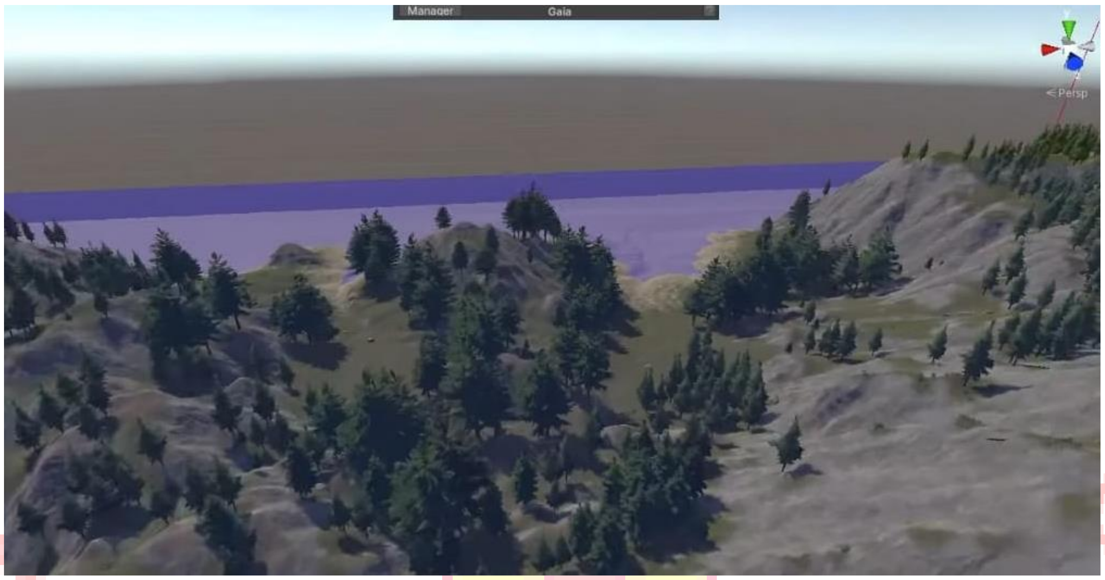
Test and iterate: Test your VR game on the target platform to ensure it functions correctly. Check for any performance issues, bugs, or usability problems. Iterate on your design based on user feedback to improve the gameplay experience.

Optimize performance: VR games require high performance to maintain a smooth and immersive experience. Optimize your game by minimizing the number of draw calls, using efficient shaders, and optimizing the use of resources. Consider implementing occlusion culling and level of detail techniques to optimize rendering performance.

Build and deploy: Once you are satisfied with your VR game, build it for the target platform. For example, build and deploy it on Oculus Quest, HTC Vive, or PlayStation VR. Unity provides options to build and deploy to different VR platforms.

Publish and distribute: If you want to make your VR game available to the public, publish it on app stores (e.g., Oculus Store, Steam) or distribute it through other channels. Follow the respective guidelines and requirements for each platform to submit your game for review and distribution.

Output:



Result:

Thus VR game was developed using Unity software .