Department of Computer Engineering

Academic Year: 2022-23 (Term I)

LAB MANUAL

(ACADEMIC RECORD)

NAME OF THE SUBJECT: Computer Graphics

CLASS: SE Div A

SEMESTER: III

Program Outcomes as defined by NBA (PO)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review 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 the public health and safety, and the 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 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.

Institute Vision : To create value - based technocrats to fit in the world of work and research

Institute Mission : To adapt the best practices for creating competent human beings
to work in the world of technology and research.

Department Vision : To provide an intellectually stimulating environment for education, technological excellence in computer engineering field and professional training along with human values.

Department Mission :

M 1: To promote an educational environment that combines academics with intellectual curiosity.

M2: To develop human resource with sound knowledge of theory and practical in the discipline of Computer Engineering and the ability to apply the knowledge to the benefit of society at large.

M3: To assimilate creative research and new technologies in order to facilitate students to be a lifelong learner who will contribute positively to the economic well-being of the nation.

Program Educational Objectives (PEO):

PEO1: To explicate optimal solutions through application of innovative computer science techniques that aid towards betterment of society.

PEO2: To adapt recent emerging technologies for enhancing their career opportunity prospects.

PEO3: To effectively communicate and collaborate as a member or leader in a team to manage multidisciplinary projects

PEO4: To prepare graduates to involve in research, higher studies or to become entrepreneurs in long run.

Program Specific Outcomes (PSO):

PSO1 : To apply basic and advanced computational and logical skills to provide solutions to computer engineering problems

PSO2 : Ability to apply standard practices and strategies in design and development of software and hardware based systems and adapt to evolutionary changes in computing to meet the challenges of the future.

PSO3 : To develop an approach for lifelong learning and utilize multi-disciplinary knowledge required for satisfying industry or global requirements.

Course Name: Computer Graphics Lab (R-19)

Course Code: CSL 303

Year of Study: S.E., Semester: III

Course Outcomes

CSC303.1	Understand the basic concepts of Computer Graphics.
CSC303.2	Demonstrate various algorithms for basic graphics Primitives.
CSC303.3	Apply 2-D geometric transformations on graphical objects.
CSC303.4	Use various Clipping algorithms on graphical objects.
CSC303.5	Explore 3-D geometric transformations, curve representation techniques and projections methods.
CSC303.6	Explain visible surface detection techniques and Animation.

C E R T I F I C A T E

This is to certify that Mr. / Miss CHANDARKAR AKSHAY RAMESH of

S.E Class COMP-A Roll No. 70

Subject COMPUTER GRAPHICS has performed the experiments /

Sheets mentioned in the index, in the premises of this institution.

Practical Incharge	**Head of Dept.**	**Principal**

Date

Examined on

Examiner 1 **Examiner 2**

Experiment No	Name of the Experiment	CO covered	Page no.	Date	Signature
1	To implement Digital Differential Analyzer line drawing algorithm.	CSC303.2	7-11		
2	To implement line using Bresenham's Line Drawing Algorithm.	CSC303.2	12-15		
3	To implement Area Filling Algorithm.	CSC303.2	16-18		
4	To apply the basic 2D Transformations such as Translation, Scaling, Rotation for a given 2D objects.	CSC303.3	19-27		
5	To implement Bezier Curve.	CSC303.5	28-31		
6	To implement program for projection of 3D object on Projection Plane.	CSC303.5	32-36		
7	Mini Project / Case Study	CSC303.1 CSC303.2 CSC303.3 CSC303.4 CSC303.5 CSC303.6			

EXPERIMENT NO. 1

Date of Performance :

Date of Submission :

Aim: To implement DDA Algorithm for drawing a line segment between two given end points A (x1, y1)and B(x2, y2)..

SOFTWARE USED: TurboC

THEORY:

DDA algorithm is an incremental scan conversion method. The characteristic of the DDA algorithm is to take unit steps along one coordinate and compute the corresponding values along

the other coordinate. Its faster method for calculating position than direct use of equation

$y = mx + c$.

$\Delta y = m\Delta x$

It slope is less than or equal to 1, we shape simple at unit x intervals(x=1) and compute each successive 'y' values as Yk+1=Yk+in

'k' subscript takes int values starting from 1 for the 1_{st} point increases by 1 unit final endpoint is reached. Since 'm' can be any read number from 0 to 1, calculated 'y' values must be rounded to nearest integers. For lines with a positive slope greater than 1 we required reverses the rows of 'x' and 'y' .That is we sample at unity intervals (y =1).

Calculate each succeeding 'x' value as Xk+1= Xk+1/ m

This is based on the assumption that lines are to be processed from left end point to right end point to reverse processing direction, either we have

x=-1, Yk+1=Yk-m

OR

y=-1, Xk+1=Xk-1/m

It absolute value of slope is less than 1 start end point is at left, we get x=1 calculate 'y' value such equation Yk+1=Yk+m.

It start end point is at right we set x=-1 equation 'y' passing from equation

Yk+1=Yk-m

ALGORITHM:

1) Read the line end points (x1, y1)and (x2, y2)
2) Calculate dx = x2 –x1 and dy = y2-y1
3) If $dx \geq dy$ then Length = dx

Else Length = dy

4) dx = x2-x1/length dy = y2- y1/length

5) x = x1 + 0.5 sign(dx) y = y1 + 0.5 sign(dy)

7) I = 1

While ($I \leq$ length)

{Plot integer (x), integer(y)

X = x + dx ; Y = y + dy ; I = ++}

8) Stop

ADVANTAGES OF DDA ALGORITHM

1. It is the simplest algorithm and it does not require special skills for implementation.

2. It is a faster method for calculating pixel positions than the direct use of equation y = mx + b.

3. 3.It eliminates the multiplication in the equation by making use of raster characteristics

DISADVANTAGES OF DDAALGORITHM

1. Floating point arithmetic in DDA algorithm is still time-consuming.

2. The algorithm is orientation dependent. Hence end point accuracy is poor.

CONCLUSION: Thus successfully Implemented simplest implementation for lines, the DDA algorithm interpolates values in interval by computing for each x_i the equations $x_i = x_{i-1} + 1$, $y_i = y_{i-1} + m$, where $\Delta x = x_{end} - x_{start}$ and $\Delta y = y_{end} - y_{start}$ and $m = \Delta y/\Delta x$

SOURCE CODE:

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
void main()
{
float x,y,x1,y1,x2,y2,dx,dy,length;
int i,gd,gm;
clrscr();
/* Read two end points of line
--------------------------------- */
printf("Enter the value of x1 :\t");
scanf("%f",&x1);
printf("Enter the value of y1 :\t");
scanf("%f",&y1);
printf("Enter the value of x2 :\t");
scanf("%f",&x2);
printf("Enter the value of y2 :\t");
scanf("%f",&y2);
/* Initialise graphics mode
--------------------------------- */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"\\tc\\BGI");
dx=abs(x2-x1);
dy=abs(y2-y1);
if (dx >= dy)
{
length = dx;
}
else
{
length = dy;
}
```

```
dx = (x2-x1)/length;
dy = (y2-y1)/length;
x = x1 + 0.5; /* Factor 0.5 is added to round the values */
y = y1 + 0.5; /* Factor 0.5 is added to round the values */
putpixel (x, y, 15);
i = 1; /* Initialise loop counter */
while(i <= length)
 {
 x = x + dx;
 y = y + dy;
putpixel (x, y, 15);
i = i + 1;
delay(100); /* Delay is purposely inserted to see
 observe the line drawing
process */
}
getch();
closegraph();
}
```

OUTPUT:

SIGN AND REMARK:

DATE:

EXPERIMENT NO.2

Date of Performance :

Date of Submission:

AIM: To implement Bresenham's line drawing algorithm for drawing a line segment between two given endpoints A (x1, y2) and B(x2, y2).

SOFTWARE AND HARDWARE REQUIRED: TurboC

THEORY:

Bresenham's Line Drawing Algorithm uses only integer addition and subtraction and multiplication by 2 and we know that the computer can perform the operations of integer addition and subtraction very rapidly. To accomplish this algorithm always increments either x or y by one unit depending on the slope of line. The increment in the other variable is determined by examining the distance between the actual line location and the nearest pixel. This distance is called as decision variable or the error.

Mathematical terms: Error or Decision variable is defined as e=Db-Da or Da-Db

Let us define e=Db-Da. Now if e>0 then it implies that Db>Da

i.e., the pixel above the line is closer to the true line.If Db<Da (i.e.,e<0) then we can say that the pixel below the line is closer to the true line.

Thus by checking only the sign of error term it is possible to determine the better pixel to represent the line path.

The error term is initially set as $e = 2\Delta y - \Delta x$

where $\Delta y = y2-y1$ and $\Delta x = x2-x1$

Then according to the value of e, following actions are taken while (e>=0)

{ y=y+1 ; e=e-2* Δx }

x = x + 1

$e = e + 2\Delta y$

While e>=0

Error is initialized with $e = e - 2\Delta y$.

ALGORITHM:

Bresenham's Line Algorithm for $(m= \Delta y/ \Delta x)/<1$

1. Read line end points (x_1,y_1) and (x_2,y_2) such that they are not equal

[If they are equal then plot the points and exit].

2. $\Delta x=|x2-x1|$ and $\Delta y=|y2-y1|$.

3. [Initialize starting point] x=x1, y=y1

4. plot(x,y);

5. $e = 2*\Delta y - \Delta x$

[Initialize value of decision variable or error to compensate for non-zero intercepts].

6. i=1 [Initialize counter]

7. while(e>=0)

{y=y+1

$e = e - 2\Delta x\}x$

= x+1

$e = e + 2*\Delta y$

8. Plot(x,y)

8. i = i+1

9. If ($i<=\Delta x$) then go to step 6

10. Stop.

BRESENHAMS LINE DRAWING ALGORITHM ADVANTAGES :

- Arithmetic- uses fixed points i.e. Integer Arithmetic.
- Operations- uses only subtraction and addition in its operations.
- Speed- Bresenhams algorithm is faster than DDA algorithm in line drawing because it performs only addition and subtraction in its calculation and uses only integer arithmetic so it runs significantly faster.
- Accuracy- more efficient and much accurate than DDA algorithm.
- Round Off- does not round off but takes the incremental value in its operation.
- Expensive- less expensive than DDA algorithm as it uses only addition and subtraction

CONCLUSION: Thus successfully implemented the Bresenham's line drawing algorithm, in which next pixel should be selected in order to form a close approximation to a straight line between two points.

SOURCE CODE:

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
#include<conio.h>
void main()
{
        float x,y,x1,y1,x2,y2,dx,dy,e;
        int i,gd,gm;
        clrscr();

/* Read two end points of line
---------------------------------- */
printf("Enter the value of x1 :\t");
scanf("%f",&x1);
printf("Enter the value of y1 :\t");
scanf("%f",&y1);
printf("Enter the value of x2 :\t");
scanf("%f",&x2);
printf("Enter the value of y2 :\t");
scanf("%f",&y2);
/* Initialise graphics mode
---------------------------------- */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"c:\\tc\\BGI");

dx=abs(x2-x1);
dy=abs(y2-y1);

/* Initialise starting point
----------------------------*/
x = x1;
y = y1;
putpixel (x, y, 15) ;

/* Initialise decision variable
-------------------------------- */
e = 2 * dy-dx;

i = 1;    /* Initialise loop counter */
do
{
while(e >= 0)
        {
        y = y + 1;
        e = e - 2 * dx;
        }
        x = x + 1;
        e = e + 2 * dy;
        i = i + 1;
        putpixel (x, y, 15);
}
while( i <= dx);
```

```
getch();
closegraph();
}
```

OUTPUT:

SIGN AND REMARK

DATE:

EXPERIMENT NO. 3

Date of Performance:

Date of Submission :

AIM: To implement area filling algorithm.

SOFTWARE REQUIRED: Turbo C.

THEORY:

BOUNDARIES FILL ALGORITHM:-

The boundary fill algorithm works as its name. This algorithm picks a point inside an object and starts to fill until it hits the boundary of the object. The color of the boundary and the color that we fill should be different for this algorithm to work.

- Start at a point inside a region
- Paint the interior outward toward the boundary
- The boundary is specified in a single color
- Fill the 4−connected or 8−connected region

```
void boundaryFill4 (int x, int y, int fill, int boundary)
{ int current; current = getPixel (x,y);
if (current != boundary && current !=fill)
{ setColor(fill);
setPixel(x,y);
boundaryFill4 (x+1, y, fill, boundary);
boundaryFill4 (x−1, y, fill, boundary);
boundaryFill4(x, y+1, fill, boundary);
bonddaryFill4(x, y−1, fill, boundary); } }
```

FLOOD FILL ALGORITHM:-

Sometimes we come across an object where we want to fill the area and its boundary with different colors. We can paint such objects with a specified interior color instead of searching for particular boundary color as in boundary filling algorithm.
Instead of relying on the boundary of the object, it relies on the fill color. In other words, it replaces the interior color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed.

- Used when an area defined with multiple color boundaries
- Start at a point inside a region
- Replace a specified interior color (old color) with fill color
- Fill the 4−connected or 8−connected region until all interior points being replaced

```
void floodFill4
(int x, int y, int fill, intoldColor)
{
if (getPixel(x,y) == oldColor)
{ setColor(fill);
setPixel(x,y);
```

```
floodFill4 (x+1, y, fill, oldColor);
floodFill4 (x−1, y, fill, oldColor);
floodFill4(x, y+1, fill, oldColor);
floodFill4(x, y−1, fill, oldColor); } }
```

CONCLUSION: Thus successfully Implemented Polygon fill algorithms Flood Fill Algorithm Boundary Fill Algorithm

SOURCE CODE:

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

void flood(int, int, int, int);
void main()
{int gd,gm;

/* Initialise graphics mode
--------------------------------- */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"\\tc\\bgi");

rectangle(50,50,100,100);
flood(55,55,4,15);
getch();
closegraph();
}

void flood(int seed_x,int seed_y,int foreground_col,int background_col)
{
if(getpixel(seed_x,seed_y)!= background_col &&
   getpixel(seed_x,seed_y)!= foreground_col)
  {
   putpixel(seed_x,seed_y,foreground_col);
   flood(seed_x+1,seed_y,foreground_col,background_col);
   flood(seed_x-1,seed_y,foreground_col,background_col);
   flood(seed_x,seed_y+1,foreground_col,background_col);
   flood(seed_x,seed_y-1,foreground_col,background_col);
   flood(seed_x+1,seed_y+1,foreground_col,background_col);
   flood(seed_x-1,seed_y-1,foreground_col,background_col);
   flood(seed_x+1,seed_y-1,foreground_col,background_col);
   flood(seed_x-1,seed_y+1,foreground_col,background_col);
  }
}
```

OUTPUT:

SIGN AND REMARK

DATE:

EXPERIMENT NO. 4

Date of Performance :

Date of Submission :

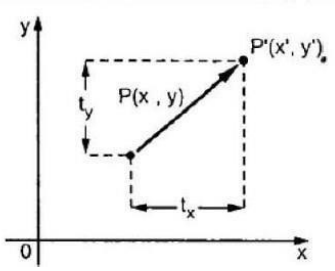
AIM: To apply the basic 2D transformations such as translation, Scaling, Rotation for a given 2D object.

SOFTWARE REQUIRED: TurboC

THEORY:

TRANSLATION: -

It is a process of changing the position of an object in a straight line path from one co-ordinate location to another. We can translate a two dimensional points by adding translation distance tx,ty to original co-ordinate position (x,y) to move point to new position (x,y) to (x',y') as shown in figure.



x' = x + tx

y' = y + ty

The translation distance vector point (tx,ty) is called a translation vector or shift vector.
It is possible to express translation equation in a single matrix equation as a column vector to represent co-ordinate positions and translation vector or shift vector.
It is possible to express translation equation as a single matrix using column vector to represent
co-ordinate position and translation vector.

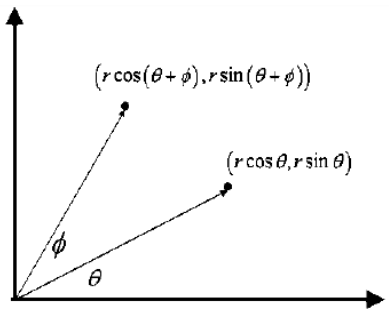
$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad T = \begin{bmatrix} tx \\ ty \end{bmatrix}$$

This allows us to write two dimensional translation equations to matrix form.
P' = P + T

ROTATION:-

A two dimensional rotation is applied to an object by repositioning it along a circular path in x-y plane.

To generate a rotation we specify rotation angle _ and position of rotation.



x' = rcos(Φ+Θ)
y' = rsin (Φ+Θ)

The equation for rotation is

x' = rcos(Φ+Θ) + rsin (Φ+Θ)
y' = - rsin (Φ+Θ) + rcos(Φ+Θ)

The rotation matrix can be given as

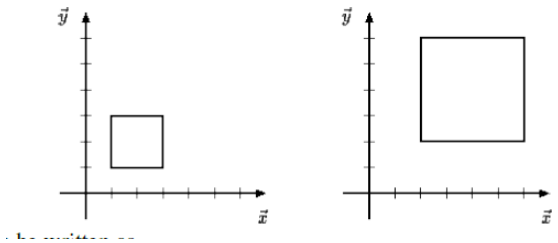
R = cosΘ sinΘ
-sinΘ cosΘ

Homogeneous co-ordinate for rotation R = cosΘ sinΘ 0
-sinΘ cosΘ 0
0 0 1

SCALING:-

A scaling transformation changes size of an object scaling factor Sx scale object in xdirection Sy in y-direction.

x' = x.Sx y' = y.Sy



The equation can be written as

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix}$$

Homogeneous co-ordinate for scaling. S =

$$\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} * [s]$$

ALGORITHM:

TRANSLATION: -

1. Read vertices of polygon.
2. Read translation vector tx and ty.
3. For i = 0 to vertex * 2
Add tx and ty to each polygon vertex.

4. End for.
5. Draw polygon.
6. Stop.

ROTATION: -

1. Read vertices of polygon.
2. Read angle of rotation.
3. For i = to vertex * 2

Poly[i] = Poly[i]cosΘ - Poly[i+1]sinΘ

Poly[i] = Poly[i]sinΘ - Poly[i+1]cos.Θ

4. Draw polygon.
5. Stop.

CONCLUSION: Successfully Implemented basic 2D transformations such as translation, Scaling, Rotation for a given 2D object.

SOURCE CODE:

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

  int main()
  {
  int i, x, y, tx, ty, sx, sy, angle=10, xmax, ymax, xmid, ymid, op;
  int gd,gm;
  float p1[10]=  {       50,50,
                 100,50,
                 100,100,
                 50,100,
                 50,50,
           };
  int pi[10];
  float b[3][3]={        1,0,0,
                 0,1,0,
                 0,0,1
         };
  int c[1][1];
  float a[1][1];
  printf("\nSelect the transformation : ");
  printf("\n1 : Traslation");
  printf("\n2 : Rotation");
  printf("\n3 : Scaling");
  printf("\n4 : Rotation about arbitrary point");
  printf( "\nEnter the option : ");
  scanf("%d",&op);
  switch(op)
  {
        case 1: printf("\nEnter x traslation : ");
                       scanf("%d",&tx);
                       printf("\nEnter y traslation : ");
                       scanf("%d",&ty);
```

```
        b[0][0] = 1;
        b[0][1] = 0;
        b[0][2] = 0;

        b[1][0] = 0;
        b[1][1] = 1;
        b[1][2] = 0;

        b[2][0] = tx;
        b[2][1] = ty;
        b[2][2] = 1;

        break;
case 2:     printf("\nEnter Rotation angle : ");
                scanf("%d",&angle);
        b[0][0] =cos(angle*3.142/180);
        b[0][1] =sin(angle*3.142/180);
        b[0][2] = 0;

        b[1][0] =-sin(angle*3.142/180);
        b[1][1] = cos(angle*3.142/180);
        b[1][2] = 0;

        b[2][0] = 0;
        b[2][1] = 0;
        b[2][2] = 1;

        break;
case 3:     printf("\nEnter x scaling : ");
        scanf("%d",&sx);
        printf("\nEnter y scaling : ");
        scanf("%d",&sy);

        b[0][0] = sx;
        b[0][1] = 0;
        b[0][2] = 0;

        b[1][0] = 0;
        b[1][1] = sy;
        b[1][2] = 0;

        b[2][0] = 0;
        b[2][1] = 0;
        b[2][2] = 1;

        break;
case 4:     printf("\nEnter x coordinate of arbitrary point : ");
        scanf("%d",&x);
        printf("\nEnter y coordinate of arbitrary point : ");
        scanf("%d",&y);
        printf("\nEnter Rotation angle : ");
        scanf("%d",&angle);
```

```
        tx = x;
        ty = y;

        b[0][0] =cos(angle*3.142/180);
        b[0][1] =sin(angle*3.142/180);
        b[0][2] = 0;

        b[1][0] =-sin(angle*3.142/180);
        b[1][1] = cos(angle*3.142/180);
        b[1][2] = 0;

        b[2][0] = -tx* cos(angle*3.142/180) + ty*sin(angle*3.142/180)+tx;
        b[2][1] = -tx* sin(angle*3.142/180) - ty*cos(angle*3.142/180)+ty;
        b[2][2] = 1;

 }
        detectgraph(&gd,&gm);
        initgraph(&gd,&gm,"\\tc\\bgi");       // Initialize graphics
xmax = getmaxx();            // Get maximum x coordinate
ymax = getmaxy();            // Get maximum y coordinate
xmid = xmax/2;               // Get the center x coordinate
ymid = ymax/2;               // Get the center y coordinate

setcolor(1);
line(xmid,0,xmid,ymax);      // Draw y coordinate
line(0, ymid, xmax, ymid);   // Draw x coordinate

 setcolor(4);
 for (i=0; i<8;i=i+2)
 {
 line(p1[i]+xmid,ymid-p1[i+1],xmid+p1[i+2],ymid-p1[i+3]);
 }
 for(i=0;i<9;i=i+2)
 {       a[0][0]=p1[i];
        a[0][1]=p1[i+1];
        c[0][0] = a[0][0]*b[0][0]+a[0][1]*b[1][0]+b[2][0];
        c[0][1] = a[0][0]*b[0][1]+a[0][1]*b[1][1]+b[2][1];
        pi[i]=c[0][0];
        pi[i+1]=c[0][1];
 }
setcolor(15);
 for (i=0; i<8;i=i+2)
 {
 line(xmid+pi[i],ymid-pi[i+1],xmid+pi[i+2],ymid-pi[i+3]);
 }
 getch();
 closegraph();
 return 0;
 }
```

OUTPUT:

01.TRANSLATION

02.ROTATION

03.SCALING

04.ROTATION ABOUT ARBITRARY POINT

SIGN AND REMARK
DATE:

EXPERIMENT NO. 5

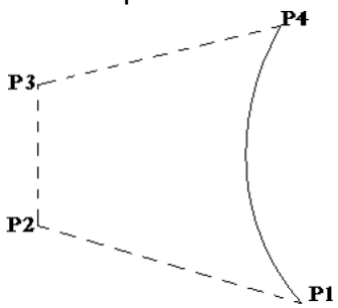
Date of Performance :

Date of Submission :

AIM: To implement Bezier curve.

SOFTWARE REQUIRED: Turbo C.

THEORY:
A Bezier curve is determined by defining polygon. They have a compiler no of properties that make them lengthy useful & convenient for curve and surface design. Picture above shows the Bezier curve and its four control point.



Bezier curve begin at the first control pts and ends at the fourth. This means if we want to connect two Bezier curve. We have to make first control point of the second curve to match the last control point of first curve. To observe line connecting1st and second control points. Similarly at the end of the curve, the curve is tangent to the arc connecting third and fourth points. We have to place the third and fourth control point at the first curve on the same line specified by this and second control point g curve.

The equations for Bezier curve are as follows:

$X = x_4a_3 + 3x_3a_2(1-a) + 3x_2a(1-a)_2 + x_1(1-a)_3$

$Y = y_4a_3 + 3y_3a_2(1-a) + 3y_2a(1-a)_2 + y_1(1-a)_3$

$Z = z_4a_3 + 3z_3a_2(1-a) + 3z_2a(1-a)_2 + z_1(1-a)$

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 that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3.
- 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=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

ALGORITHM:

1. Start
2. Read no g control points as n.
3. Read the control point as P(x,y)
4. Draw the convex polygon by joining two points.
5. The Bezier formation is calculated using the point as $P(1)=\Sigma P(k)\, B_k\, n(4)$
   Where n varies from 0.001 to 1 be small interval
   $B(k)=n(u) = (n,k) - u_k(1-u)_{n-k}$ and (n,k) = n! / (k! (n-k)!)
6. Plot the point (x,y)
7. u=u+0.01
8. repeat step 5 & 7 until u<1
9. stop

CONCLUSION- Successfully implemented cubic Bezier curve using 4 given control points.

SOURCE CODE:

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<stdio.h>
#include<process.h>
int gd,gm,maxx,maxy;
float xxx[4][2];

/* Function to draw line from relative position
        specified in array xxx----------------------*/

void line1(float x2,float y2)
{
line(xxx[0][0],xxx[0][1],x2,y2);
xxx[0][0]=x2;
xxx[0][1]=y2;
}
/* Bezier function
-------------------- */
void bezier(float xb,float yb,float xc,float yc,float xd,float yd,int n)
{
float xab,yab,xbc,ybc,xcd,ycd;
float xabc,yabc,xbcd,ybcd;
float xabcd,yabcd;
if (n==0)
        {
         line1(xb,yb);
         line1(xc,yc);
         line1(xd,yd);
        }
else
```

```
        {
         xab = (xxx[0][0]+xb)/2;
         yab = (xxx[0][1]+yb)/2;
         xbc = (xb+xc)/2;
         ybc = (yb+yc)/2;
         xcd = (xc+xd)/2;
         ycd = (yc+yd)/2;
         xabc = (xab+xbc)/2;
         yabc = (yab+ybc)/2;
         xbcd = (xbc+xcd)/2;
         ybcd = (ybc+ycd)/2;
         xabcd = (xabc+xbcd)/2;
         yabcd = (yabc+ybcd)/2;
         n=n-1;
         bezier(xab,yab,xabc,yabc,xabcd,yabcd,n);
         bezier(xbcd,ybcd,xcd,ycd,xd,yd,n);
        }
}

/* Function to initialise graphics
--------------------------------- */
void igraph()
        {
        detectgraph(&gd,&gm);
        if(gd<0)
                {
                puts("CANNOT DETECT A GRAPHICS CARD");
                exit(1);
                }
        initgraph(&gd,&gm,"\\tc\\bgi");
        }
    void  main()
        {
        int i;
        float temp1,temp2;
        igraph();

        /* Read two end points and two control points of the curve
        ---------------------------------------------------------- */
        for(i=0;i<4;i++)
        {
        printf("Enter (x,y) coordinates of point%d : ",i+1);
        scanf("%f,%f",&temp1,&temp2);
        xxx[i][0] = temp1;
        xxx[i][1] = temp2;
        }
        bezier(xxx[1][0],xxx[1][1],xxx[2][0],xxx[2][1],xxx[3][0],xxx[3][1],8);
        getch();
        closegraph();
}
```

OUTPUT:

SIGN AND REMARK

DATE:

EXPERIMENT NO. 6

Date of Performance :

Date of Submission :

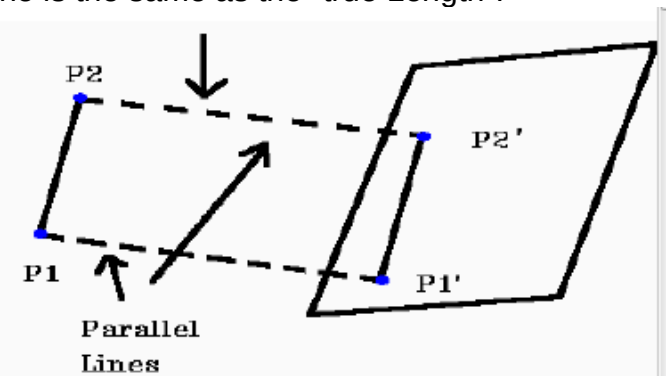
AIM: To implement program for projection of 3D object on projection plane.

SOFTWARE USED: TurboC

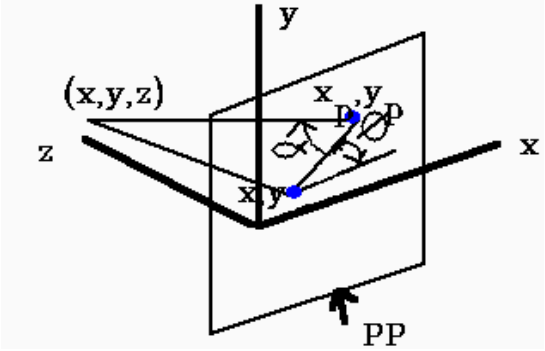
THEORY:

PARALLEL VIEWING PROJECTIONS

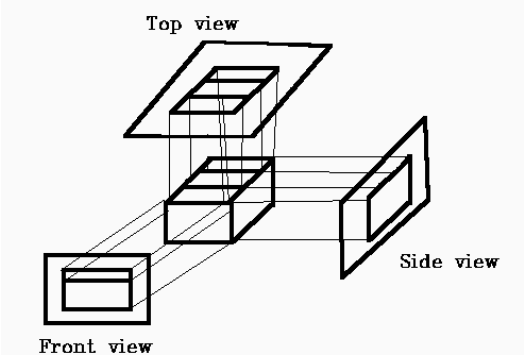
Projection rays (projectors) emanate from a Center of Projection (COP) and intersect Projection Plane (PP). The COP for parallel projectors is at infinity. The length of a line on the projection plane is the same as the "true Length".



There are two different types of parallel projections:
If the direction of projection is perpendicular to the projection plane then it is an orthographic projection. If the direction of projection is not perpendicular to the projection plane then it is an oblique projection.
Look at the parallel projection of a point (x, y, z). (Note the left handed coordinate system). The projection plane is at z = 0. x, y are the orthographic projection values and xp, yp are the oblique projection values (at angle a with the projection plane)



Look at orthographic projection: it is simple, just discard the z coordinates. Engineering drawings frequently use front, side, top orthographic views of an object. Here are three orthographic views of an object.



Orthographic projections that show more than 1 side of an object are called axonometric orthographic projections. The most common axonometric projection is an isometric projection where the projection plane intersects each coordinate axis in the model coordinate system at an equal distance.

ISOMETRIC PROJECTION
The projection plane intersects the x, y, z axes at equal distances and the projection plane Normal
makes an equal angle with the three axes.
To form an orthographic projection xp = x, yp= y ,zp = 0. To form different types e.g., Isometric,
just manipulate object with 3D transformations.

OBLIQUE PROJECTION

The projectors are not perpendicular to the projection plane but are parallel from the object to the
projection plane The projectors are defined by two angles A and d where:

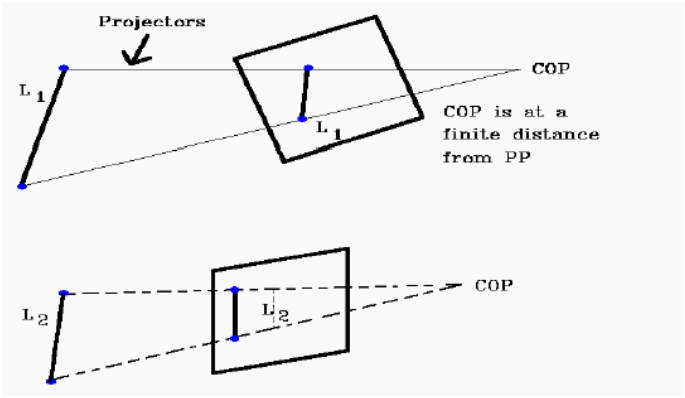
A = angle of line (x,y,xp,yp) with projection plane,
d = angle of line (x, y, xp, yp) with x axis in projection plane
L = Length of Line (x,y,xp,yp).
Then:
cos d = (xp - x) / L ------ >xp = x + Lcos d ,
sin d = (yp - y) / L ------ >yp = y + Lsin d ,
tan A = z / L

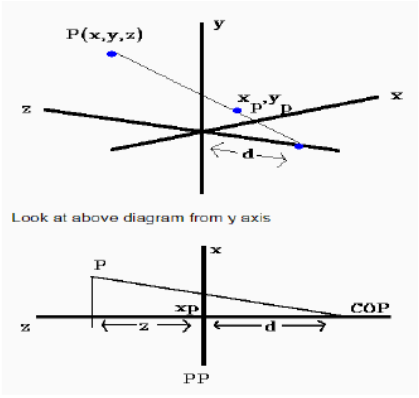
Now define L1 = L / z ---- > L= L1 z ,
so tan A = z / L = 1 / L1 ; xp = x + z(L1cos d) ; yp = y + z(L1sin d)

$$P = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ L1\cos q & L1\sin q & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

PERSPECTIVE VIEWING PROJECTION
The Perspective viewing projection has a Center of Projection ("eye") at a finite distance from
the projection plane (PP).
So the distance of a line from the projection plane determines its size on the projection plane, i.e.
the farther the line is from the projection plane, the smaller its image on the projection plane.
In

the two images above, the projections of L1 = L2 but the actual length of L1 <> L2.
Perspective
projection is more realistic since distant objects appear smaller.
Computing the Perspective Projection





Now x / (z+d) = xp/d
xp = x[d / (z+d)]
xp = x / (z / d + 1)
Do same for y (look down the x axis) and get
y p = y / (z / d+1)
zp = 0
Note that we can increase the perspective effect by decreasing d (moving closer). We can represent this in matrix form by using homogeneuos coordinates as follows:

[xhyhzh w] = [x y z 1] |1 0 0 0 |
|0 1 0 0 |
|0 0 0 1/d|
|0 0 0 1 |

where:
xh = x . yh= y, zh = 0 ,w = (z/d) + 1
And Points on the projection plane are [xpypzp 1] = [xh/w yh/w zh/w 1]
This leads to the same xp, yp as before.

CONCLUSION: Successfully implemented program for projection of 3D given object.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include<graphics.h>
#include<conio.h>
void draw3d(int s,int x[20],int y[20],int d);
void main()
{
 int gd=DETECT,gm;
 int x[20],y[20],i,s,d;
 initgraph(&gd,&gm,"C:\\TC\\BGI");
 printf("Enter the No of sides : ");
 scanf("%d",&s);
 for(i=0;i<s;i++)
 {
          printf("(x%d,y%d) :",i,i);
          scanf("%d%d",&x[i],&y[i]);
 }
 printf("Depth :");
 scanf("%d",&d);
 draw3d(s,x,y,d);
 getch();
 setcolor(14);
 for(i=0;i<s-1;i++)
 {
          line(x[i]+200,y[i],x[i+1]+200,y[i+1]);
 }
 line(x[i]+200,y[i],x[0]+200,y[0]);
 getch();//top view
 for(i=0;i<s-1;i++)
 {
          line(x[i],300,x[i+1],300);
          line(x[i],300+d*2,x[i+1],300+d*2);
          line(x[i],300,x[i],300+d*2);
          line(x[i+1],300,x[i+1],300+d*2);
 }
 getch();//side view
 for(i=0;i<s-1;i++)
 {
          line(10,y[i],10,y[i+1]);
          line(10+d*2,y[i],10+d*2,y[i+1]);
          line(10,y[i],10+d*2,y[i]);
          line(10,y[i+1],10+d*2,y[i+1]);
 }
 getch();
 closegraph();
}
void draw3d(int s,int x[20],int y[20],int d)
{
 int i,j,k=0;
```

```
for(j=0;j<2;j++)
 {
        for(i=0;i<s-1;i++)
              line(x[i]+k,y[i]-k,x[i+1]+k,y[i+1]-k);
        line(x[i]+k,y[i]-k,x[0]+k,y[0]-k);
        k=d;
 }
 for(i=0;i<s;i++)
        line(x[i],y[i],x[i]+d,y[i]-d);
}
```

OUTPUT:

SIGN AND REMARK

DATE: