SIES GST, NERUL, NAVI MUMBAI

DEPARTMENT OF COMPUTER ENGINEERING

LAB MANUAL

Computer Graphics(CSL402)

B.E. (COMPUTER ENGINEERING)

SEMESTER-IV

(R-2016)-Ver1

COMPUTER ENGINEERING DEPARTMENT

DEPARTMENT'S VISION

To be a centre of Excellence in Computer Engineering to fulfill the rapidly growing needs of the Society.

DEPARTMENT'S MISSION

M1: To Impart quality education to meet the professional challenges in the area of Computer Engineering.

M2: To create an environment for research, innovation, professional and social development.

M3: To nurture lifelong learning skills for achieving professional growth.

M4 To strengthen the alumni and industrial interaction for overall development of students.

PROGRAMME EDUCATIONAL OBJECTIVES(PEOs)

PEO1: Practice Computer engineering in core and muti-disciplinary domains.

PEO2: Exhibit leadership skills for professional growth.

PEO3: Pursue higher Studies for career advancement

PROGRAMME OUTCOMES (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics science engineering fundamentals and an mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems engineering problems.

PO2: 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.

PO3: 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

PO4: 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.

PO5: 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.

PO6: The engineer and society: Apply reasoning informed by Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: 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 and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual and as a member or leader in diverse teams and individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: 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, and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and 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.

PO12: Life-long learning: long learning: Recognize the need for and have the Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

PROGRAMME SPECIFIC OUTCOMS (PSOs)

PSO1: To apply computational and logical skills to solve Computer engineering problems.

PSO2: To develop interdisciplinary skills and acquint with cutting edge technologies in software industries.

GENERAL INSTRUCTIONS (Do's And Don'ts)

1. Wearing ID-Card is compulsory.
2. Keep your bag at the specified place.
3. Shut down the system after use.
4. Place the chairs in proper position before leaving the laboratory.
5. Report failure/Non-working of equipment to Faculty In-charge / Technical Support staff immediately.
6. Know the location of the fire extinguisher and the FIRST-AID Box and how to use then in case of an emergency.
7. Do not eat or drink in the laboratory.
8. Do not litter in the laboratory.
9. Avoid stepping on electrical wires or any other computer cables.
10. Do not open the system unit casing or monitor casing particularly when the power in turn ON.
11. Do not insert metal objects such as clips, pins and needles into the computer casing. They may cause fire.
12. Do not remove anything from laboratory without permissions.
13. Do not touch, connect or disconnect any plug or cable without permission.

LAB OUTCOMES (LO)

LO1: Explore the working principle, utility of various input/ output devices and graphical tools
LO2: Implement various output and filled area primitive algorithms using c/ opengl.
LO3: Apply transformation on graphical objects.
LO4: Apply clipping on graphical objects.
LO5: Implementation of curve and fractal generation
LO6: Develop a graphical application based on learned concept

LAB ARTICULATION MATRIX (MAPPING WITH PO & PSO)

	Lab Outcome	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
LO1	Explore the working principle, utility of various input/ output devices and graphical tools	3													
LO2	Implement various output and filled area primitive algorithms using c/ opengl.	3	2			3	1		3	3	3				1
LO3	Apply transformation on graphical objects.	3	2						3	3	3			3	
LO4	Apply clipping on graphical objects.	3	2						3	3	3				
LO5	Implementation of curve and fractal generation		3						3	3	3				
LO6	Develop a graphical application based on learned concept.	3													

Average	3	2.25			3	1		3	3	3			3	1

Laboratory Assessment

Academic Year: 2019- 20 **Class/Sem: IV** **Div:C** **Batch**:2018-22

Student Name: ___________________________ **Roll No:** ____________________

Course Name: ____________________________ **Course Code**: _______________

1. Preparedness and Efforts/ Preparation and Knowledge		
3: Well prepared and puts efforts.	2: Not prepared but puts efforts or prepared but doesn't put efforts	1: Neither prepared nor puts efforts
2. Presentation of output/ Accuracy and Neatness of Documentation		
3: Uses all perfect Instructions / Interrupts/Presented well.	2: Uses some perfect Instructions / Interrupts/ moderate presentation.	1: Doesn't use any of the proper Instruction / Interrupt/ Not presented properly.
3. Results/ Participation in Practical Performance		
3: Participate and gets proper results	2: Participate but doesn't get proper result or gets result but with the help of faculty in-charge	1: Neither Participate nor gets the results
4. Punctuality		
3: Get the experiment checked in-time and is always in-time to the lab sessions	2: Some time delays the experiment checking or is late to the lab sessions for few times	1: Most of the time delays experiment checking and / or comes late for lab sessions
5. Lab Ethics		
3: Follows proper lab ethics by keeping the lab clean and placing things at their right place	2: Sometimes doesn't follow the lab ethics	1: Most of the times makes the lab untidy and keeps things at wrong place

Performance Indicator/ Expt. No → ↓	1	2	3	4	5	6	7	8	9	10	11	Total
	LO2	LO2	LO 2	LO 2	LO 3	LO 4	LO2	LO4	LO5	LO 5	LO1	
1. Preparedness and Efforts/ Preparation and Knowledge												
2. Presentation of output/ Accuracy and Neatness of Documentation												
3. Debugging and results/ Participation in Practical Performance												
4. Punctuality												
5. Lab Ethics												
Total												
Average												

EVALUATION:

Exceed Expectations (3), Meet Expectations (2). Below Expectations (1)

Faculty In-charge

LIST OF EXPERIMENTS

SR NO	EXPERIMENT TITLE
1	Implement Line Drawing Algorithm. - DDA Line drawing Algo. and Bresesnham's Line Drawing Algorithm.
2	Implement Mid-point circle drawing algorithm
3	Implement Mid-Point ellipse drawing algorithm.
4	Implementing Boundary Fill and Flood Fill algorithm.
5	Implement 2D Transformations.
6	Implement Liang Barsky Line Clipping Algorithm.
7	Implement Character generation method.
8	Implement Sutherland Hodgeman Polygon Clipping Algorithm.
9	Implement Bezier Curve.
10	Implement Koch curve.
11	Implement Basic Primitives using Opengl.

PROGRAM NO. 1: Line Drawing Algorithm

Aim: - To implement Line Drawing Algorithms.

I) DDA (Digital Differential Analyzer)
II) Bresenham's Line Algorithm

Theory: - The DDA is a scan conversion line algorithm based on calculating either dy or dx. We sample the line at unit intervals either in one coordinate and describe corresponding integer values nearest the line path for the other coordinates. An accurate and efficient faster line generating algorithm developed by Bresenhams can convert line only using incremental integer calculations that can be adopted to display circles and other curves sampling at unit x intervals we need to decide which of the two possible pixel positions is closer to line path.

Algorithm:

DDA Line Algorithm:

```
1. Input x1, y1, x2, y2, dx, dy, steps and k are of type integer and xinc, yinc, x, y of type
   float.
2. dx = x2-x1
   dy = y2-y1

3. if (abs(dx) > abs (dy))
           steps= dx;

   else steps = dy;

4. xinc = dx / steps;
5. yinc = dy / steps


7. x=x1;

   y=y1;

  set pixel (round(x),round(y));

8. for k=1 to steps
   x = x + xinc;

   y= y + yinc;

   set pixel(round(x),round(y));

9. STOP.
```

Bresenham’s Line Drawing Algorithm:

1. Accept x1, y1, x2, y2.
2. Calculate dx, dy, 2dx,2dy,2dy-2dx;
3. Find Initial Decision parameter as p = 2* dy – dx;
4. At each xk along the line starting at k=0 perform the following test
   If (pk<0) next point to be plot is (xk+1,yk) & pk+1= pk+2*dy

   Else next point is (xk+1, yk+1) and

   Pk+1= pk + 2*dy – 2*dx;

5. Repeat the above step till k<dx;

Conclusion: DDA Line drawing algo. Contains floating point arithmetic and round up operation. Bresenham’s Line Drawing algo. Contains only integer arithmetic.

PROGRAM NO. 2: Circle Drawing Algorithm

Aim: To implement Mid Point circle drawing algorithm.

Theory:-

As in the raster line algorithm we sample at unit intervals & determine the closest pixel position to the specified circle path for each step. For a given radius r and screen center position (xc,yc). We can first setup our algorithm to calculate pixel position around a circle path centered at coordinate origin(0,0). Then each calculated position (x,y) is moved to its proper position by adding xc to x & yc to y. Positions in other seven octants are obtained later by using symmetry.

Algorithm For Mid Point Circle:

1. Input radius r and circle centre (xc, yc) and obtain the first point on circle centered on origin (0,r).
2. Calculate Initial Decision parameter as p = (5/4) – r.
3. At each xk position starting at k=0 perform the following test:
   If(pk<0)

   Then next point is (xk+1, yk).

   pk+1 = pk + 2xk+1 +1

   Else

   Next point is (xk+1, yk+1)

   pk+1 = pk + 2xk+2 +2yk – 2

4. Determine symmetrical points in other seven octants.

5. Repeat above steps until x>=y.

Conclusion: Mid Point Circle drawing algo use Symmetry concept. Circle is symmetric about Octant and Circle function act as a decision parameter.

PROGRAM NO. 3: Ellipse Drawing Algorithm

Aim: - To implement Mid Point Ellipse Drawing Algorithm.

Theory: -
Mid point Ellipse drawing algorithm uses quad symmetry of an ellipse.

Equation of an ellipse is:

$((X^2 / (rx^2)) + ((y^2) / (ry^2))$

An ellipse is considered to consist of two regions R1 & R2. The slope at the intersection point of these regions is +1 or -1.

Slope is calculated as,

$dy / dx = -ry^2x / rx^2y$

Algorithm:

1. Input rx, ry and ellipse center (xc,yc), and obtain the first point on an ellipse centered on the origin as

    $(x_0, y_0) = (0, r_y)$

2. Calculate the initial value of the decision parameter in region 1 as

    $p1_0 = ry^2 – rx^2\ ry + ¼\ rx^2$

3. At each xk position in region, starting at k = 0, perform the following test:

    If p1k<0, the next point along the ellipse centered on (0,0) is (xk+1,yk) and

        p1k+1 = p1k + 2ry2xk+1 + ry2

    Otherwise, the next point along the circle is (xk+1, yk-1) and

        $p1k+1 = p1k + 2ry^2xk+1 – 2ry^2yk+1 + ry^2$

  With $2ry^2xk+1 = 2ry^2xk + 2ry^2$, $2rx^2yk+1 = 2rx^2\ yk – 2rx^2$

And continue until 2ry2x >= 2rx2 y.

4. Calculate the initial value of the decision parameter in region 2 using the last point (x0, y0) calculated in region 1 as

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

5. At each yk position in region2, starting at k = 0, perform the following test:

    If p2k>0, the next point along the ellipse centered on (0,0) is (xk, yk -1) and
    P2k+1 = p2k – 2rx2yk+1 + rx2
    Otherwise, the next point along the circle is (xk+1, yk-1) and

P2k+1 = p2k – 2ry2xk+1 + rx2

Using the same incremental calculations for x and y as region1.

6. Dete4rmine symmetry points in the other three quadrants.

7. Move each calculated pixel position (x, y) onto the elliptical path centered on (xc, yc) and plot the coordinate values:

x = x + xc,   y = y + yc

8. Repeat the steps for region1 until $2\ ry^2x >= 2rx^2y$

Conclusion:
Mid Point ellipse drawing implemented using concept of symmetry and for region1 and region2 pixel positions are calculated.

PROGRAM NO. 4: Polygon Filling Methods

Aim: - To implement Boundary and Flood Fill Algorithms.

Theory: -

Boundary Fill Algorithm:
This algorithm is very simple. It needs one seed point which is surely inside the polygon. This is called "Seed Point". The algorithm checks to see if the seed pixel has a boundary pixel color or not.

In this method, edges of the polygon are drawn. Then starting with some seed, any point inside the polygon we examine the neighboring pixels to check whether the process is continued until boundary pixels are reached. Since this process involves checking of boundaries. This method is also called Boundary Fill Method.

Boundary defined regions may be either 4 connected or 8 connected. If a region is connected, then every pixel in the region may be reached by a combination of moves in only four directions: left, right, up and down. For an 8 connected region every pixel in the region may be reached by a combination of moves in the two horizontal, two vertical and four diagonal directions.

In some cases, an 8 connected is more accurate than the 4 connected.

The following procedure illustrates the recursive method for filling a 4 connected region with color specified in parameter fill color (f_color) up to boundary color specified with parameter boundary color (b_color).

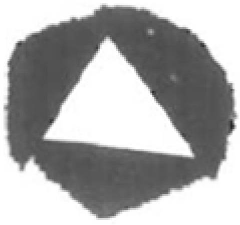
Algorithm:

```
Void boundary_fill(int x,int y, int fcolor,int bcolor)
{

if((getpixel(x,y) != fcolor) && (getpixel(x,y) != bcolor))
{
putpixel(x,y,fcolor);
boundary_fill(x+1,y,fcolor,bcolor);
boundary_fill(x-1,y,fcolor,bcolor);
boundary_fill(x,y-1,fcolor,bcolor);
boundary_fill(x,y+1,fcolor,bcolor);
boundary_fill(x+1,y-1,fcolor,bcolor);
boundary_fill(x+1,y+1,fcolor,bcolor);
boundary_fill(x-1,y-1,fcolor,bcolor);
boundary_fill(x-1,y+1,fcolor,bcolor);
}
}
```

Flood Fill Algorithm:

Sometimes we want to fill in (or recolor) an area that is not defined within a single color boundary. Figure below shows an area bordered by several different color regions.



An area defined within multiple color boundaries.

We can paint such areas by replacing a specified interior color instead of searching for a boundary color value. This approach is called a flood-fill algorithm.

We start from a specified interior point (x, y) and reassign all pixel values that are currently set to a given interior color with the desired fill color.

If the area we want to paint has more than one interior color, we can first reassign pixel values so that all interior points have the same color.

Using either a 4-connected or 8-connected approach, we then step through pixel positions until all interior points have been repainted.

The following procedure flood fills a 4-connected region recursively, starting from the input position.

Flood Fill Function:

```
void floodFill4 ( int x, int y, intfillColor , intoldColor)
{
i f (getpixel (x, y) == oldcolor)
{
setcolor ( fillColor ) ;
setpixel (x, y ) :
floodFill4 (x + l, y, fillColor, oldColor):
floodfill4 (x-1, y, fillcolor, oldcolor);
floodPill4 (x, y + l, fillcolor, oldcolor);
floodFill4 (x, y-1, fillColor, oldcolor);
}
}
```

We can modify procedure floodFill4 to reduce the storage requirements of the stack by filling horizontal pixel spans.

In this approach, we stack only the beginning positions for those pixel spans having the value oldColor. Starting at the first position of each span, the pixel values are replaced until a value other than oldColor is encountered.

Conclusion:- Boundary fill algorithm is used polygon having same boundary color. Flood Fill algo. is used polygon having different color boundaries.

PROGRAM No. 5: 2D Transformations

Aim: -. Write a program to implement 2D transformations.

Theory:-

Translation:
Translation is a process of changing the position of an object from one coordinate
Location to another. We can translate a two dimensional point by adding translation distances Tx and Ty to the original coordinate position (x,y) to move the new position (x',y').

$$x' = x + Tx$$
$$y' = y + Ty$$

The translation distance pair (Tx, Ty) is called a Translation Vector or Shift Vector.
The homogeneous coordinate for translation are given as:

P' = P * T

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & Tx \\ 0 & 1 & Ty \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x + Tx \\ y + Ty \\ 1 \end{pmatrix}$$

2. Rotation:

A two dimensional rotation is applied to an object by repositioning it along a circular path in the xy plane

To generate rotation, we specify a rotation angle theta and the position of the rotation point about which the object
is to be rotated.

The transformation equation for rotating a point(x,y) through an angle theta about the origin as:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} \begin{matrix} = \\ = \end{matrix} \begin{pmatrix} x \cos\theta - y \sin\theta \\ x \sin\theta + y \cos\theta \end{pmatrix}$$

Homogeneous coordinate for Rotation are given as:

$$R = \begin{pmatrix} \text{Cos}\,\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

p' = p * R

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ 1 \end{pmatrix}$$

3. Scaling:

Scaling transformation changes the size of an object. This operation can be carried out for polygons by multiplying
the coordinate values (x,y) of each vertex by scaling factors Sx and Sy to produce the transformed coordinate (x',y').

x' = x * Sx
y' = y * Sy

Scaling factor scales the object in the x direction and scaling factor scales the object in the y direction.

Homogeneous coordinate for scaling are given as:

$$S = \begin{pmatrix} Sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Therefore

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x * Sx \\ Y * Sy \\ 1 \end{pmatrix}$$

4. Shear Transformation:

A transformation that slants the shape of an object is called the Shear Transformation. Two common shearing transformations are used.
One shifts x coordinate values and other shifts y coordinate values.

X Shear:

The x shear preserves the y coordinates, but changes the x values which causes vertical lines to tilt right or left.

The transformation matrix for x shear is given as:

$$x\_sh = \begin{pmatrix} 1 & Shx & 0 \\ & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad 0$$

$x' = x + Shx * y$
$y' = y$

Y Shear:

The Y shear preserves the x coordinates, but changes the y values which causes horizontal lines to transform into lines which slope up or down.

The transformation matrix for Y shear is given as:

$$Y\text{-}sh = \begin{pmatrix} 1 & 0 & 0 \\ Shy & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$x' = x$
$y' = y + Shy * x$

5. Reflection:

A Reflection is a transformation that produces a mirror image of an object relative to an axis of reflection.
We can choose an axis of reflection in the xy plane or perpendicular to the xy plane.

Reflection about X axis;

$x' = x$
$y' = -y$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Reflection about Y axis:

$x' = -x$
$y' = y$

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Reflection about origin:

$x' = -x$
$y' = -y$

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Reflection about line y = x:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Reflection about line y = -x

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Conclusion :- Transformations like translation, rotation, scaling, reflection about x=y and Shear X and Shear Y are implemented.

PROGRAM NO. 6: Line Clipping Algorithms

Aim: - To implement Liang Barsky line clipping algorithm

Theory: - In computer graphics, line clipping is the process of removing lines or portions of lines outside of an area of interest. Typically, any line or part thereof which is outside of the viewing area is removed.
In Liang-Barsky algorithm, we first write the point-clipping conditions in parametric form as
Each of these for inequalities can be expressed as, k = 1, 2, 3, ….. Where parameter p and q are defined as
Any line that is parallel to one of the clipping boundaries has pk = 0 for the value of k corresponding to that boundary. If, for that value of k, we also find qk < 0, then the line is completely outside the boundary and can be eliminated from further consideration. If qk >=0, the line is inside the parallel clipping boundary.
When pk <0, the infinite extension of the line proceeds from the outside to the inside of the inside of the infinite extension of this particular boundary. If pk > 0, the line proceeds from the insides to the outside. For a nonzero value of pk, we can calculate the value of u that corresponds to the point where the infinitely extended line intersects the extension of boundary k as u = qk / pk.

Algorithm:

$$x = x_0 + u(x_1 - x_0) = x_0 + u\Delta x$$

$$y = y_0 + u(y_1 - y_0) = y_0 + u\Delta y$$

A point is in the clip window, if
$x_{\min} \le x_0 + u\Delta x \le x_{\max}$
and
$y_{\min} \le y_0 + u\Delta y \le y_{\max}$,
which can be expressed as the 4 inequalities
$up_k \le q_k, \quad k = 1, 2, 3, 4$,
where
$p_1 = -\Delta x, q_1 = x_0 - x_{\min}$(left)
$p_2 = \Delta x, q_2 = x_{\max} - x_0$(right)
$p_3 = -\Delta y, q_3 = y_0 - y_{\min}$(bottom)
$p_4 = \Delta y, q_4 = y_{\max} - y_0$(top)
To compute the final line segment:
A line parallel to a clipping window edge has $p_k = 0$ for that boundary.
If for that k, $q_k < 0$, the line is completely outside and can be eliminated.
When $p_k < 0$ the line proceeds outside to inside the clip window and when $p_k > 0$, the line proceeds inside to outside.
For nonzero p_k, $u = \frac{q_k}{p_k}$ gives the intersection point.

For each line, calculate $u_1$and u_2.
a) For u_1, look at boundaries for which $p_k < 0$(outside -> in). Take $u_1$to be the largest among $\left(0, \frac{q_k}{p_k}\right)$.
b)For u_2, look at boundaries for which $p_k > 0$(inside -> out). Take $u_2$to be the minimum of $\left(1, \frac{q_k}{p_k}\right)$.
c) If $u_1 > u_2$, the line is outside and therefore rejected.

Conclusion: Liang Barsky Line Clipping algorithm implemented.

PROGRAM NO. 7: Character Generation Methods

Aim: - To implement character generation method.

Theory:-
Most of the times characters re built into the graphics display devices usually as hardware but sometimes through software.

There are basic three methods of generating characters:
Stroke Method
Bitmap Method
Starbust Method

1. Stroke Method:
- This method uses a small line segment to generate a character.
- The small series of line segments are drawn like a strokes of a pen to form a character.
- We can build our own stroke method character generator by calls to the line drawing algorithm.
- Here, it is necessary to decide which line segments are needed for each character and then drawing these segments using line drawing algorithm. We can draw characters on the display.
-The stroke method supports scaling of the character.
It does this by changing the length of line segments used for character drawing.

2. Bitmap Method:

- It is also called as dot matrix because in this method characters represented by array of dots in the matrix form.
- It is a two dimensional array having columns & rows.
- An 5 *7 array is commonly used to represent a character.
- An 7*9 & 9*13 arrays are also used.
- Higher resolution devices such as inkjet printer or laser printer may use character array that are over 100 * 100.

- Each dot in the matrix is a pixel.
- The character is placed on the screen by coping pixel values from the character array into some portion of the screen's frame buffer.
- The value of the pixel control intensity of the pixel.
- The dot pattern for character are stored in the hardware device called Character Generation Chip.
- The character generator chip accepts address for the character & gives the bit pattern for that character.
- The size of a dot is fixed so the dot matrix method does not lend itself to variable sized character.

3. Starbust Method:

- In this method a fix pattern of line segments are used to generate characters.
- There are 24 line segments.
- Out of these 24 line segments, segments required to display for particular character are highlighted.
- This method of character generation is called starbust method because of its character appearance.
- The pattern for particular character are stored in the form of 24 bit code.
- Each bit representing one line segment.
- The bit is set to 1 to highlight the line segment, otherwise set to zero.
- For e.g.: 24 bit code for

Character A is 0011 0000 0011 1100 1110 0001
Character B is 0000 0011 0000 1100 1111 0011

This method of character generation is not used. Now a days because of following disadvantages:

The 24 bits are required to represent a character. Hence more memory is required.

Require code conversion software to displat character from its 24 bits code.

Character quality is poor. It is worst for curved shape characters.

Conclusion:-
Character generation methods like Stroke method and Bitmap method are implemented.

PROGRAM NO. 9: Curve Generation

AIM: - To implement Bezier Curve.

THEORY:-

Bezier curves use a different way of specifying the curve. The cubic Bezier curve requires four sample points which completely specify the curve. The curve begins at first sample points and end at fourth sample point. If we require two connected Bezier curves, it can be achieved by six sample points where third and fourth point of first curve acts as first and second points of second curve.

Equation of Bezier Curve is:

x = x4a3+3x3 a2 (1-a) +3x2a (1-a) 2+x1(1-a) 3

y = y4a3+3y3a2 (1-a) +3y2a (1-a) 2+y1 (1-a) 3

z = z4a3+3z3a2 (1-a) +3z2a (1-a) 2+z1 (1-a) 3

Here the a value of ‘a’ moves from 0 to 1 as the curve travel from first to fourth sample point. Other method of constructing Bezier curve is by taking midpoints. By taking midpoints, we can find a point on the curve and also split the curve into two sections. We can continue to split the curve into smaller sections, until we have sections so short that they cannot be repeated by straight lines or till size of section is not greater than size of pixel.

CONCLUSION:- Cubic Bezier curve is implemented using 4 control points.

PROGRAM No. 10: Fractals.

Aim : - To implement Koch curve.

Theory:-

Given a set of n points, curve generation algorithms can generate a smooth curve passing through by them. But to draw naturally realistic object we need a rough curve. E.g. Lightning.

Dimensional D of fractal can be given by N = SD

Where S is scaling factor and N is no. Of units which make up whole object.

Algorithm:

FRACTAL SUBDIVIDE(x1, y1, z1,x2, y2, z2, s, n)

Arguments:

X1, y1, z1

X2, y2, z2

S – Offset scale factor

N – Depth of recursion

Local : xmid, ymid, zmid coordinates at which to break the line.

Conclusion:-

Koch Curve is implemented using concept of fractals.

PROGRAM No.11

Aim: Draw basic primitives using opengl Functions.

Theory:
OpenGL is the software interface to the graphic hardware.
OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer.
These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images).
OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.
OpenGL can render:
Geometric primitives such as Lines, points, polygons, etc.
Bitmaps and Images
OpenGL Primitives:
Point:
GL_POINTS:
Treats each vertex as a single point.
Vertex n defines a point n.
N points are drawn.
Sample:

```
glBegin(GLPOINTS);
glVertex2f(x1,y1);
glVertex2f(x1,y1);
glEnd();
```

Line:
GL_LINES:
Treats each pair of vertices as an independent line segment.
Vertices 2n-1 and 2n define a line n.
N/2 lines are drawn.
GL_LINE_LOOP:
Draws a connected group of line segments from the first vertex to the last, then back to the first.
Vertices n and n+1 define line n.
N lines are drawn.
Polygon:
GL_TRIANGLES:
Treats each triplet of vertices as an independent triangle.
Vertices 3n-2, 3n-1, and 3n define triangle n.
N/3 triangles are drawn.
Sample:

```
glBegin(GLTRIANGLES);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glVertex2f(x3,y3);
glEnd();
```

GL_QUADS:
Treats each group of four vertices as an independent quadrilateral.
Vertices 4n-3, 4n-2, 4n-1, and 4n define quadrilateral n.
N/4 quadrilaterals are drawn.
GL_POLYGON:
Draws a single, convex polygon.
Vertices 1 through N define this polygon.
A polygon is convex if all points on the line segment between any two points in the polygon or at the boundary of the polygon lie inside the polygon.
Symbol:

```
glBegin(GLPOLYGON);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glVertex2f(x3,y3)
glVertex2f(x4,y4);
glVertex2f(x5,y5);
glEnd();
```

Setting Up: OpenGL + GLUT + Windows + Visual Studio

This guide assume you have the following installed:
Windows 7 or 8
Visual Studio 2008, 2010, or 2012
By convention, we assume the OS is installed to the C drive and VS is installed to a folder such as "C:\Program Files (x86)\Microsoft Visual Studio 10.0″ (for 2010, 2008 maps to 9.0 and 2012 to 11.0).

Installing GLUT

The first step is to download GLUT (the OpenGl Utility Toolkit). This library provides a slew of helper functions for working with OpenGL, including setting up a window. Grab the latest binaries from here. Unzip this and copy the binaries out into the following directories. Be aware that these are 32-bit binaries. You will need to create the "gl" folder in "\Microsoft Visual Studio 10.0\VC\include\", as it doesn't already exist.

File	**Location**
glut32.dll	C:\Windows\SysWOW64\
glut.h	C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\include\gl\
glut32.lib	C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\lib\

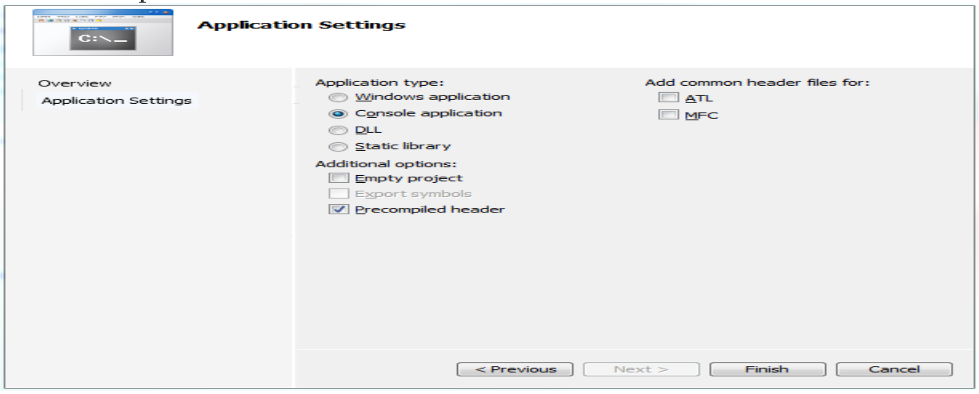
Windows 64-bit

File	**Location**
glut32.dll	C:\Windows\System32\
glut.h	C:\Program Files\Microsoft Visual Studio 10.0\VC\include\gl\
glut32.lib	C:\Program Files\Microsoft Visual Studio 10.0\VC\lib\

Windows 32-bit

Creating the VS project

Now boot up Visual Studio. You'll want to create a simple Win32 console application. In VS 2010, open the New Project dialog and look at the list of templates. Select "Visual C++\Win32\Win32 Console Application". Click through the Wizard that appears, leaving the defaults in place.



Application Settings

Overview
Application Settings

Application type:
( ) Windows application
(●) Console application
( ) DLL
( ) Static library

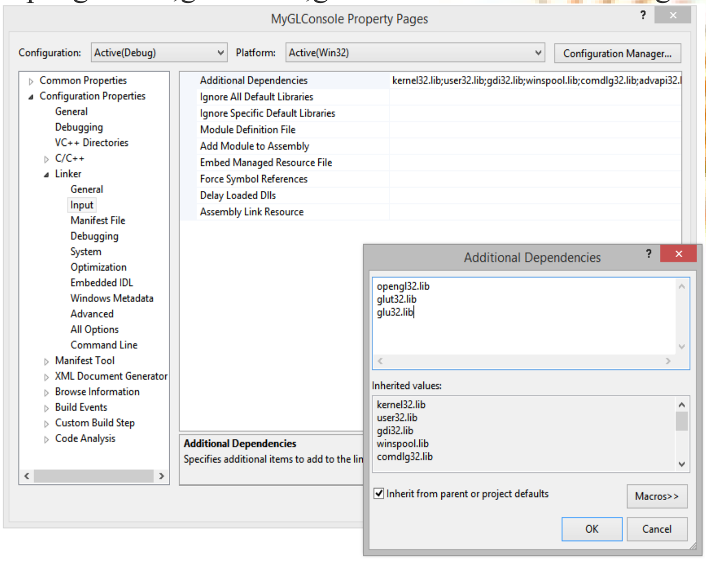
Additional options:
[ ] Empty project
[ ] Export symbols
[✓] Precompiled header

Add common header files for:
[ ] ATL
[ ] MFC

< Previous | Next > | Finish | Cancel

Linking the libraries

At this point you should get a solution with a couple of default files and a "_tmain" entry function. Right-click the project you created, go to Properties, then open Configuration Properties\Linker\Input and add the following to the "Additional Dependencies" field: "opengl32.lib;glut32.lib;glu32.lib". Close the settings.



MyGLConsole Property Pages

Configuration: Active(Debug)　Platform: Active(Win32)　Configuration Manager...

Common Properties
Configuration Properties
General
Debugging
VC++ Directories
C/C++
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Windows Metadata
Advanced
All Options
Command Line
Manifest Tool
XML Document Generator
Browse Information
Build Events
Custom Build Step
Code Analysis

Additional Dependencies	kernel32.lib;user32.lib;gdi32.lib;winspool.lib;comdlg32.lib;advapi32.l
Ignore All Default Libraries	
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the lin

Additional Dependencies

```
opengl32.lib
glut32.lib
glu32.lib
```

Inherited values:

```
kernel32.lib
user32.lib
gdi32.lib
winspool.lib
comdlg32.lib
```

[✓] Inherit from parent or project defaults

Macros>> | OK | Cancel

Conclusion: By using basic opengl functions we had implemented program for triangle.