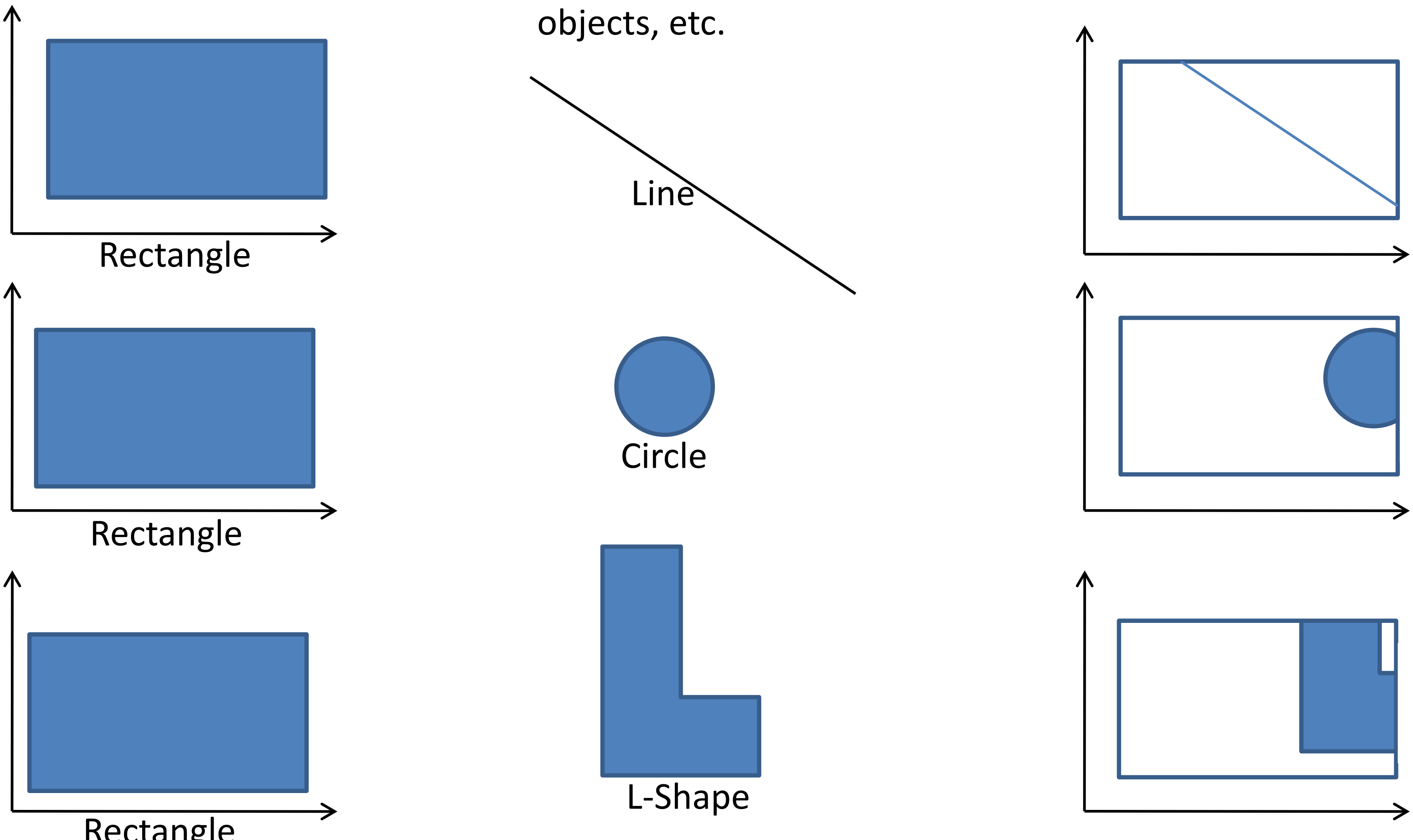
Clipping and Filling Color

Clipping

Viewport or subregion of the screen:

Drawing objects: Line, circle, conics, objects, etc.

Clipping

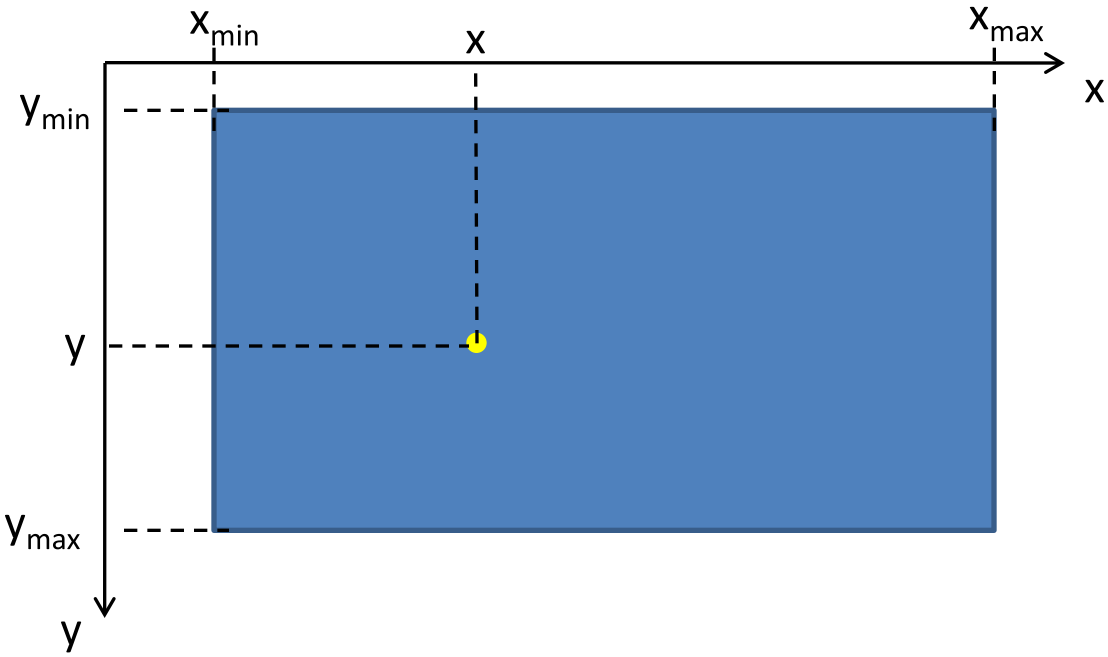


Clipping

- Clipping lines
- Clipping polygons

Clipping Endpoints

Viewport or subregion of the screen:



$$\begin{cases} x_{\min} \leq x \leq x_{\max} \\ y_{\min} \leq y \leq y_{\max} \end{cases}$$

Line Clipping

- Brute-force method:

  - Calculate the intersection points between segment line and four boundary lines of clip rectangle.

$$\begin{cases} Ax + By + C = 0 \quad \text{(segment line)} \\ A_i x + B_i y + C_i = 0, i = 1,..4 \quad \text{(rectangle edge)} \end{cases}$$

  - Check intersection point whether it lies within both the clip rectangle edge and the segment line.

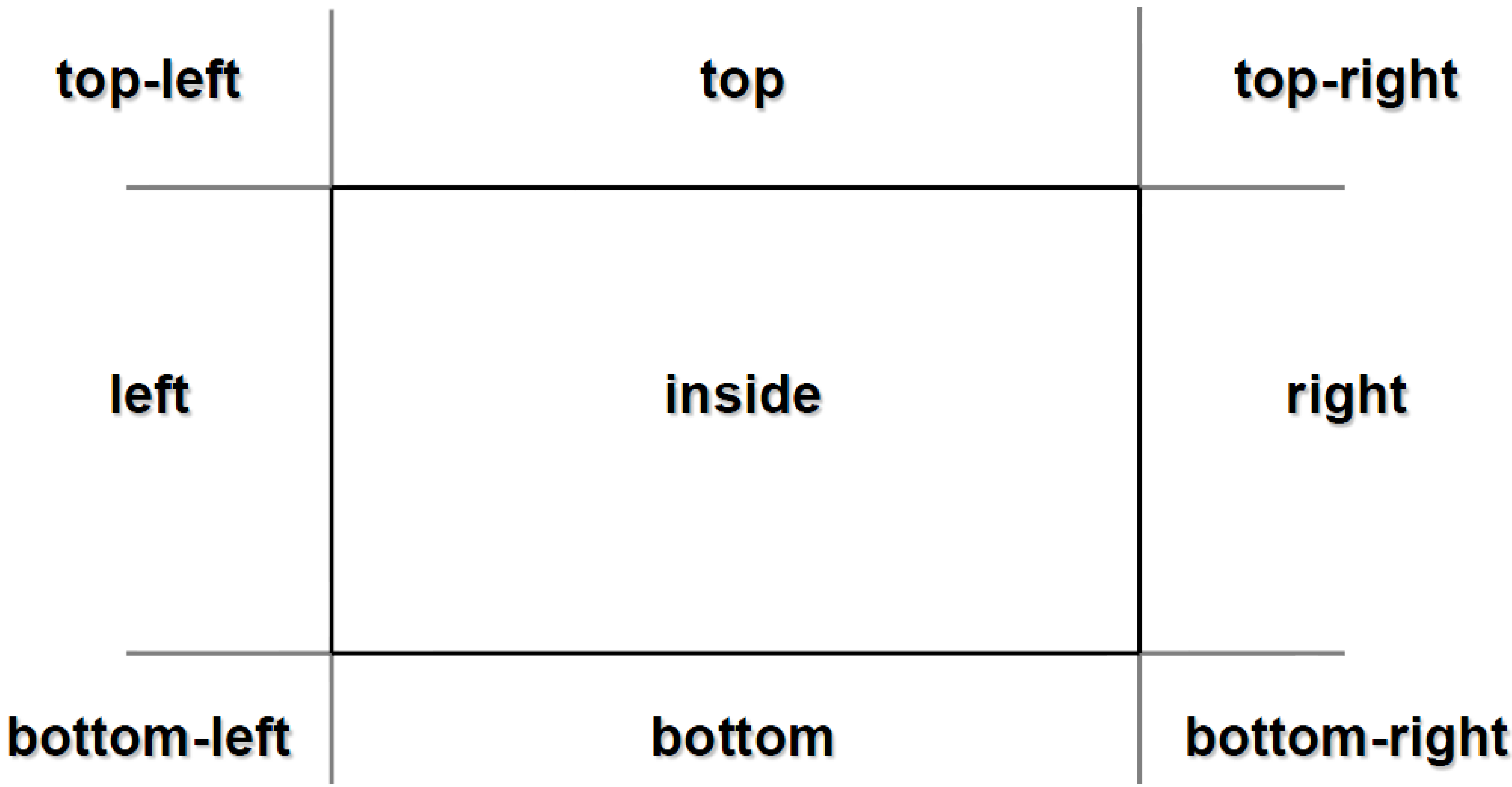
$$\begin{cases} x = x_0^i + t_{edge}(x_1^i - x_0^i) \\ y = y_0^i + t_{edge}(y_1^i - y_0^i) \end{cases}, i = 1,..,4 \quad \begin{cases} x = x_0 + t_{line}(x_1 - x_0) \\ y = y_0 + t_{line}(y_1 - y_0) \end{cases}$$

Points position between (x_0,y_0) and (x_1,y_1) iff $0 \le \{t_{line}, t_{edge}\} \le 1$

If it is, we substitute the endpoint of segment line by the intersection point and show it.

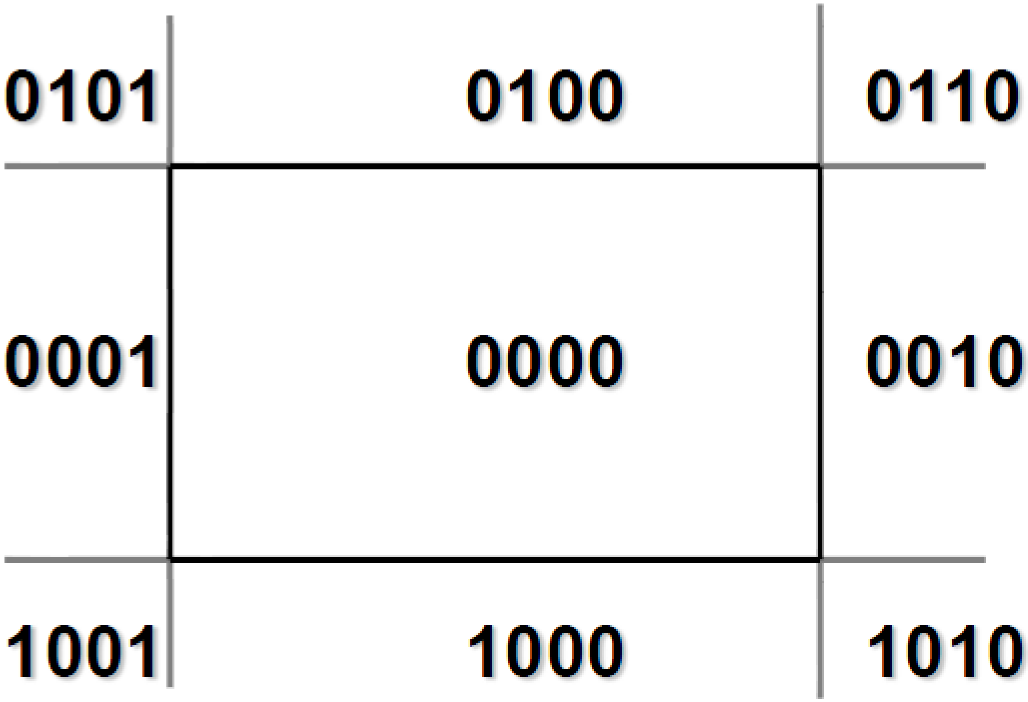
Cohen-Sutherland Line Clipping

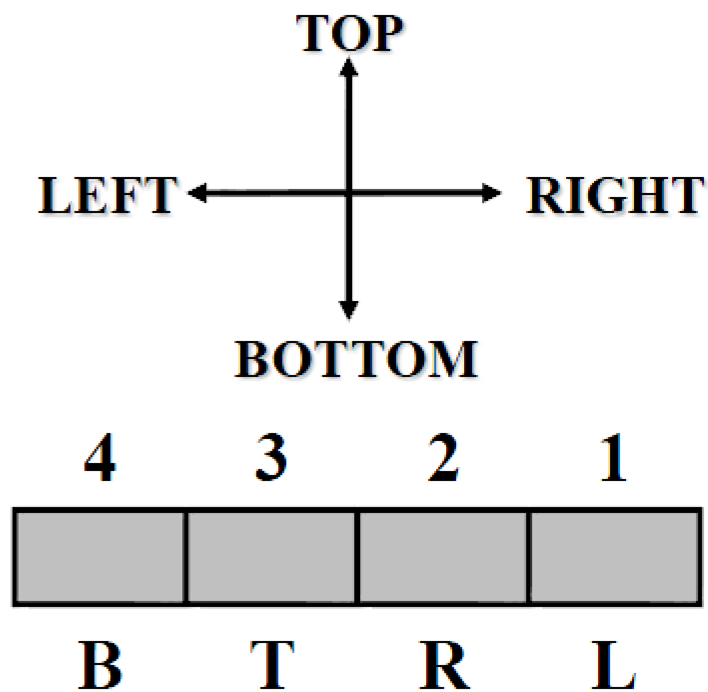
Divide to nine regions for endpoint checking into trivial acceptance and rejection



Cohen-Sutherland Line Clipping

- Encode 9 regions by 4-bit sequence





$x < x_{\min} \Rightarrow$ First bit = 1 $\quad x > x_{\max} \Rightarrow$ Second bit = 1

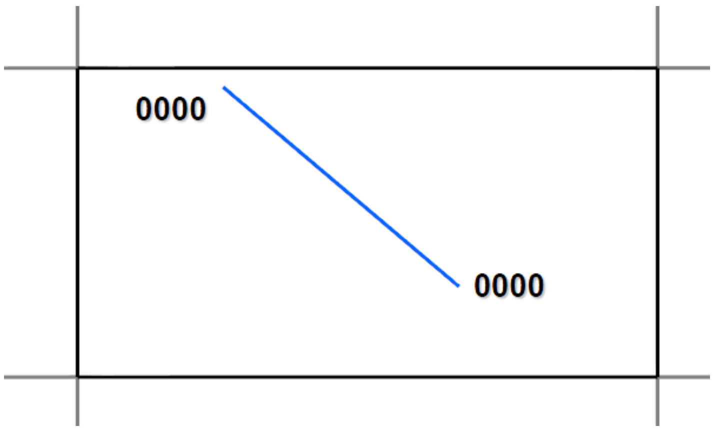
$y < y_{\min} \Rightarrow$ Third bit = 1 $\quad y > y_{\max} \Rightarrow$ Fourth bit = 1

Cohen-Sutherland Line Clipping

LEFT	0	0	0	1	= 1
RIGHT	0	0	1	0	= 2
TOP	0	1	0	0	= 4
BOTTOM	1	0	0	0	= 8

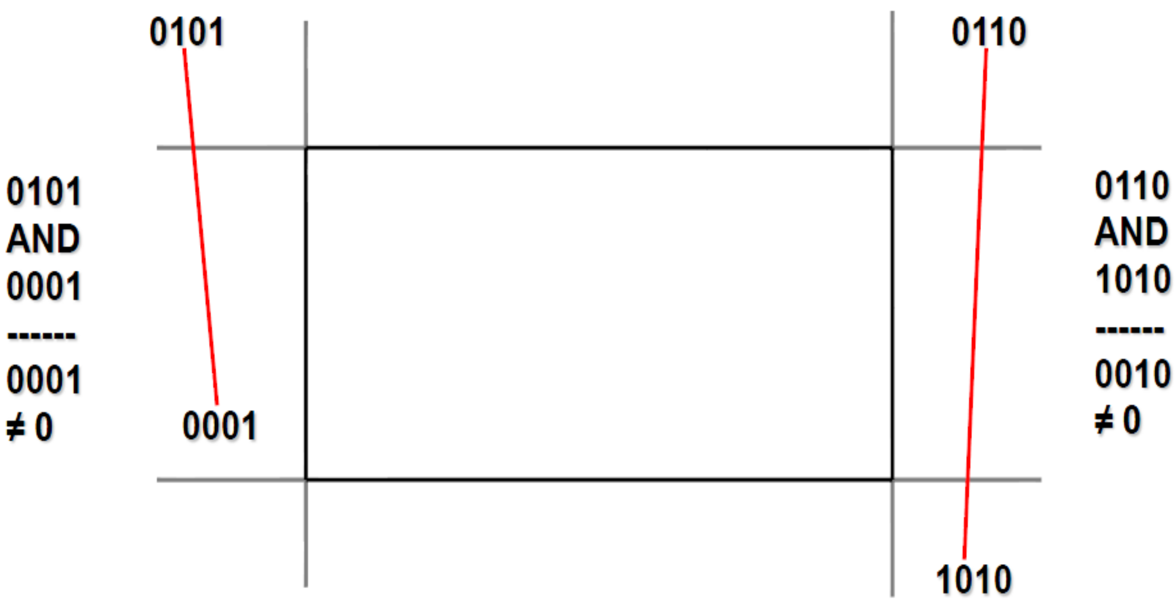
```
int Encode(Point p)
{
    int code = 0;
    if (p.x < xmin)        code |= LEFT;
    if (p.x > xmax)        code |= RIGHT;
    if (p.y > ymax)        code |= TOP;
    if (p.y < ymin)        code |= BOTTOM;
    return code;
}
```

Cohen-Sutherland Line Clipping



- Let c_1 and c_2 denote the encoding results of two endpoints.

$c_2 == c_1 == 0000$



$c_1 \text{ AND } c_2 \neq 0$

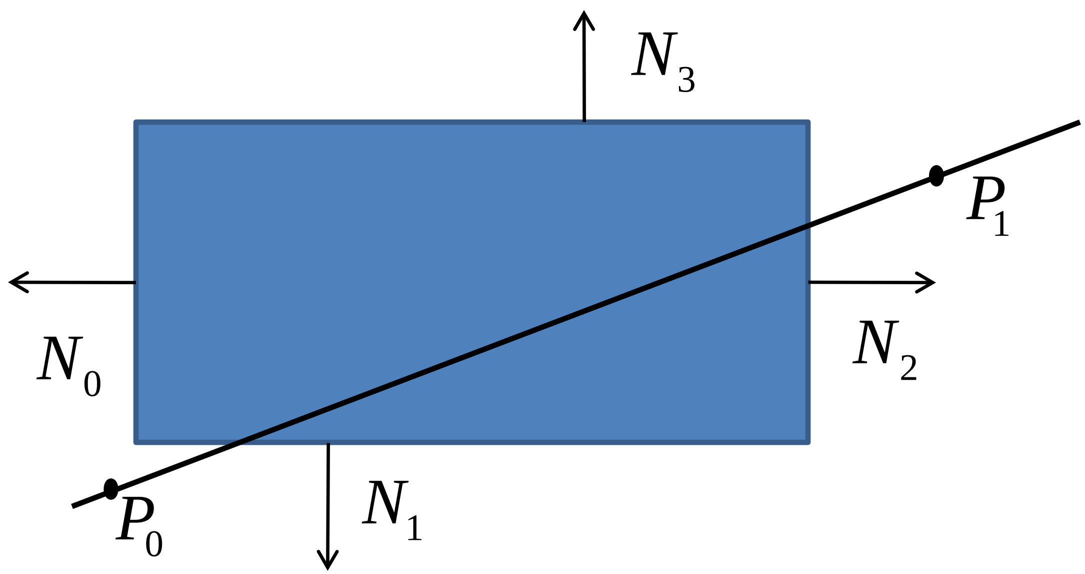
Cohen-Sutherland Line Clipping

```
c1 = Encode(endpoint1);
c2 = Encode(endpoint2);
do {
      If (c1==c2==0000)
               {Trivial acceptance; done = TRUE}
      Else if ((c1&c2) <> 0)
                {Trivial rejection; done = TRUE}
      Else {
            c = c1 ? c1 : c2;
            if  ((c & TOP) <> 0 )
               Calculate intersection point with TOP line
            else if ((c & BOTTOM) <> 0 )
               Calculate intersection point with BOTTOM line
            else if ((c &RIGHT) <> 0 )
               Calculate intersection point with RIGHT line
            else if ((c &LEFT) <> 0 )
               Calculate intersection point with LEFT line
           if (c==c1) { endpoint1 = intersection point; c1 = Encode(endpoint1); }
           if (c==c2)  { endpoint2 = intersection point; c2 = Encode(endpoint2); }
          }
} while(done==TRUE)
```

Cyrus-Beck Line Clipping

- Based on four intersections between segment line and four edge lines of viewport.
- Based on parametric equation and normal vector of the edge lines

$$P(t) = P_0 + (P_1 - P_0)t$$



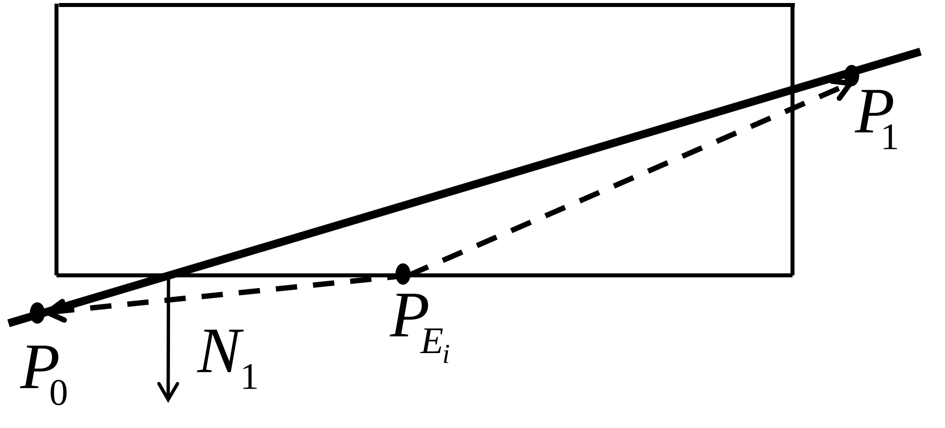
Cyrus-Beck Line Clipping

- Let $D = (P_1 - P_0)$

  If $\mathrm{N}_i . D == 0$ then goto the next case

- Check the sign of dot product

$$N_i . [P(t) - P_{E_i}] \begin{cases} > 0, & \text{potential entering PE} \\ = 0, & \text{intersection point} \\ < 0, & \text{potential leaving PL} \end{cases}$$



Cyrus-Beck Line Clipping

- For intersection, the value of t is calculated by

$$N_i.[P(t) - P_{E_i}] = 0$$

$$N_i.[P_0 - P_{E_i}] + N_i.[P_1 - P_0]t = 0$$

$$\Rightarrow t = \frac{N_i.[P_0 - P_{E_i}]}{-N_i.D} \qquad N_i D \begin{cases} < 0, \text{ potential entering PE} \\ > 0, \text{ potential leaving PL} \end{cases}$$

$$(\text{by } 0 \le t \le 1)$$

Clip edge$_i$	Normal N_i	P_{Ei}	P_0 - P_{Ei}	t
Left: $x=x_{min}$	(-1,0)	(x_{min}, y)	$(x_0 - x_{min}, y_0 - y)$	$-(x_0 - x_{min})/(x_1 - x_0)$
Right: $x=x_{max}$	(1,0)	(x_{max}, y)	$(x_0 - x_{max}, y_0 - y)$	$-(x_0 - x_{max})/(x_1 - x_0)$
Bottom: $y=y_{max}$	(0,-1)	(x, y_{max})	$(x_0 - x, y_0 - y_{max})$	$-(y_0 - y_{max})/(y_1 - y_0)$
Top: $y=y_{min}$	(0,1)	(x, y_{min})	$(x_0 - x, y_0 - y_{min})$	$-(y_0 - y_{min})/(y_1 - y_0)$

Cyrus-Beck Line Clipping

```
Calculate  N_i  and select P_Ei
If (P_1 == P_0)
   exit;
else {  tE = 0;  tL = 1;
         for i=1:4   //  4 candidates of intersection points
         {     if  (N_i.D !=0)   {
                   calculate t by using Table;
                   check sign(N_i.D) to classify as PE or PL
                   if (PE)  tE = max (tE,t);
                   if (PL)   tL = min (tL,t);
               }  // if  (N_i.D !=0)
         } // for i=1:4
         if  (tE > tL)    exit;
         else {    P_0 =  P(tE);
                   P_1 =  P(tL);
         }
} // If (P_1 == P_0) else
```

Liang-Barsky Algorithm

- Given segment line defined by P and Q
- Line is inside the clip region for value of *t* such that

$$\begin{cases} x_{\min} \le x_P + tDx \le x_{\max} \\ y_{\min} \le y_P + tDy \le y_{\max} \\ 0 \le t \le 1 \end{cases} \Rightarrow \begin{cases} p_k t \le q_k, & k = 1,2,3,4 \\ 0 \le t \le 1 \end{cases}$$

where

$$p_1 = -Dx, \quad q_1 = x_P - x_{\min}, \text{ and } (Dx, Dy) = [\text{Q-P}]$$

$$p_2 = Dx, \quad q_2 = x_{\max} - x_P$$

$$p_3 = -Dy, \quad q_3 = y_P - y_{\min}$$

$$p_4 = Dy, \quad q_4 = y_{\max} - y_P$$

Liang-Barsky Algorithm

Solve system of inequalities

$$\begin{cases} p_k t \leq q_k, & k = 1,2,3,4 \\ 0 \leq t \leq 1 \end{cases}$$

If $\exists \; k \in \{1,2,3,4\} : (p_k = 0) \wedge (q_k < 0)$, then no solution

If $\forall \; k \in \{1, 2, 3, 4\} \; : (p_k \neq 0)$, then we have

- $p_k < 0$, $t \geq q_k/p_k$, as *t* increases, line goes from outside to inside.
- $p_k > 0$, $t \leq q_k/p_k$, line goes from inside to outside.

Liang-Barsky Algorithm

Set $t_{min} = 0$ and $t_{max} = 1$

Calculates t values by q_k/p_k

- if $t < t_{min}$ or $t > t_{max}$, then exit
- Otherwise classify the t value as "potential entering" or "potential leaving" and update t_1 and t_2

$$\begin{cases} t_1 = \max(\left\{ 0 \le t = \dfrac{q_k}{p_k} \le 1, p_k < 0 \right\}, \mathrm{t}_{\min}) \\ t_2 = \min(\left\{ 0 \le t = \dfrac{q_k}{p_k} \le 1, p_k > 0 \right\}, \mathrm{t}_{\max}) \\ t_1 \le t_2 \end{cases}$$

$$Q_1(x_1 + t_1 Dx, y_1 + t_1 Dy), Q_2(x_1 + t_2 Dx, y_1 + t_2 Dy)$$

Example of Liang-Barsky Algorithm

$t_{\min} = 0, \; t_{\max} = 1$

$(Dx, Dy) = [Q - P] = (15 - (-5), 9 - 3) = (20, 6)$

$\overline{t_1} = q_1 / p_1 = (x_P - x_{\min}) / -Dx = (-5) / -20 = 1/4,$

$\Rightarrow \; p_1 = -20 < 0 \Rightarrow t_1 = \max(\overline{t_1}, t_{\min}) = 1/4$

$t_{\min} = 1/4$

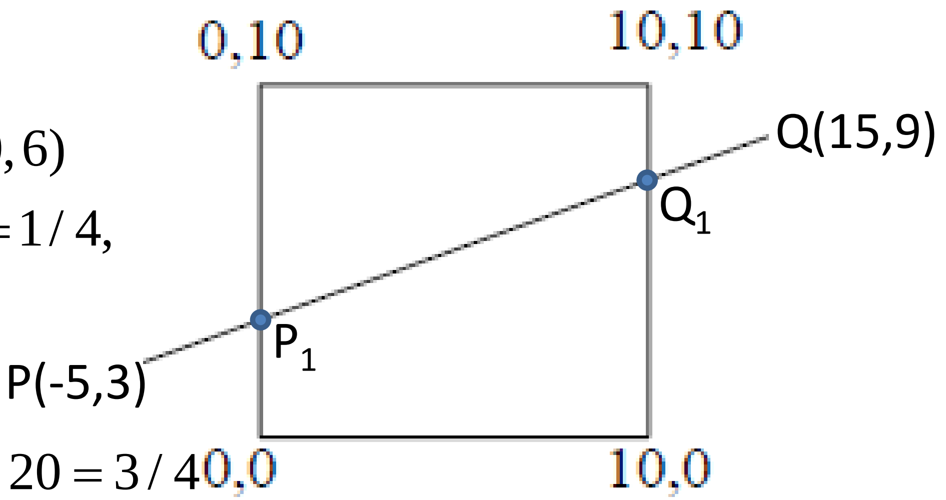
$\overline{t_2} = q_2 / p_2 = (x_{\max} - x_P) / Dx = (10 - (-5)) / 20 = 3/4$

$\Rightarrow \; p_2 = 20 > 0 \Rightarrow t_2 = \min(\overline{t_2}, t_{\max}) = 3/4$

$t_{\max} = 3/4$

$\overline{t_3} = q_3 / p_3 = (y_P - y_{\min}) / -Dy = 3/(-6) = -1/2 \quad (< 0, \text{ unchecked})$

$\overline{t_4} = q_4 / p_4 = (y_{\max} - y_P) / Dy = (10 - 3)/6 = 7/6 \quad (> 1, \text{ unchecked})$

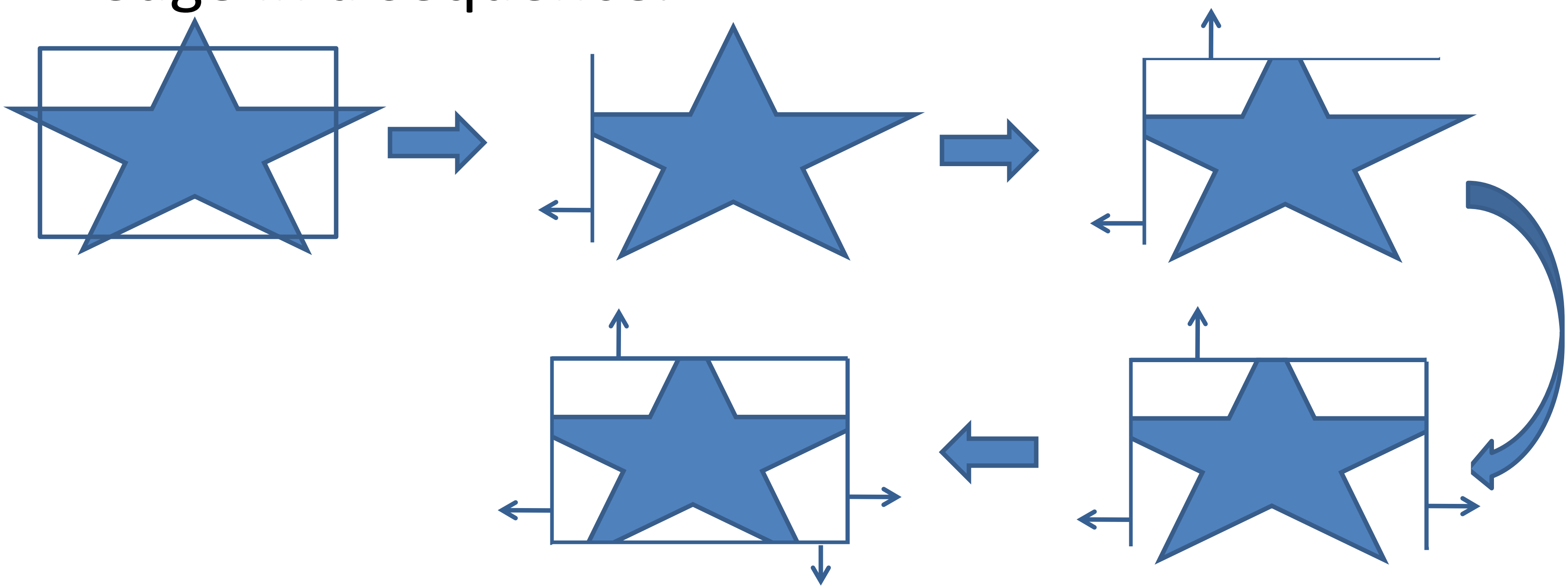


$t_1 < t_2$

P1(-5+20 x 0.25 , 3+6 x 0.25) Q1(-5+ 20 x 0.75 , 3 + 6 x 0.75)

Polygon Clipping-Sutherland-Hodgeman Algorithm

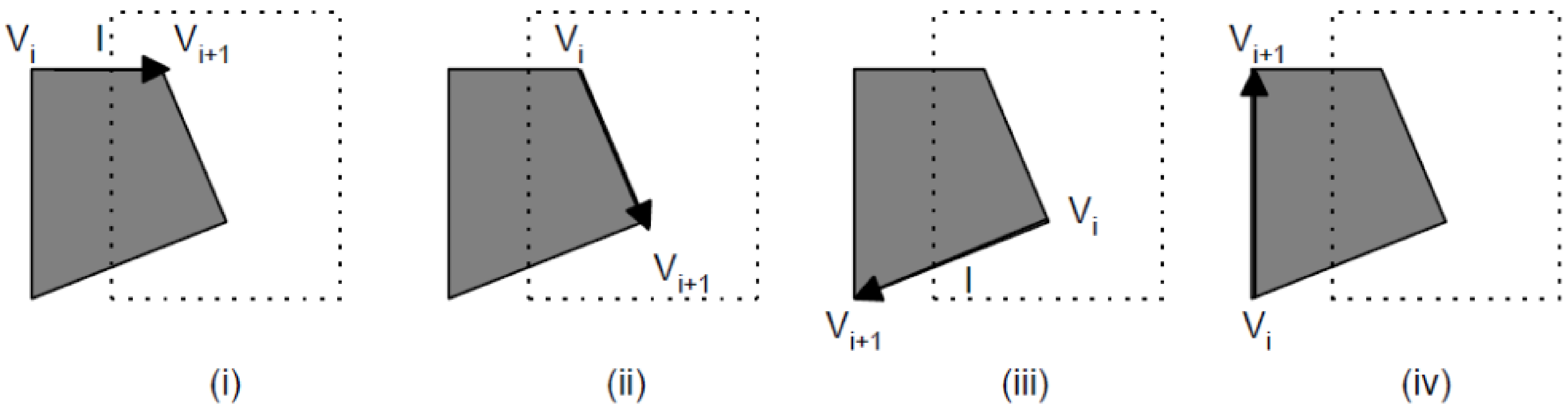
- Problem of clipping polygon: one polygon can be split into multiple polygons.
- Sutherland-Hodgeman algorithm will clip one polygon by clipping against each clip-rectangle edge in a sequence.



Polygon Clipping-Sutherland-Hodgeman Algorithm

- In each polygon-edge clipping, there are 0,1, or 2 vertices added on the output list.

- There are four possible cases

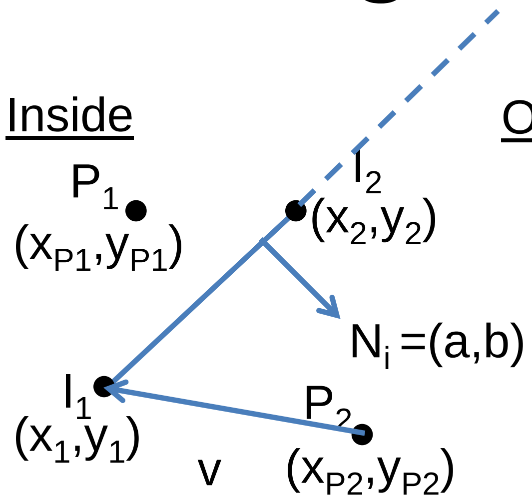


Save I and V_{i+1} | Save V_i and V_{i+1} | Save V_i and I | Save nothing

Polygon Clipping-Sutherland-Hodgeman Algorithm

```
vertexArray outputList = PolygonVertexArray;
vertexArray inputList;
for (each clip-rectangle-edge )
    inputList = outputList;
    outputList.clear();
    Point S = inputList.last;
    for (each point E in inputList)
        if (E inside clip-rectangle-edge) then
            if (S not inside clip-rectangle-edge) then
                outputList.add(ComputeIntersection(S,E,clip-rectangle-edge));
            end
            outputList.add(E);
        else if (S inside clip-rectangle-edge) then
                outputList.add(ComputeIntersection(S,E,clip-rectangle-edge));
        end
        S = E;
    end
end
```

Polygon Clipping-Sutherland-Hodgeman Algorithm



$$v = [I_1 - P_2]$$

$$N.v = |N|.|v|\cos(N_i, v)$$

$$\Rightarrow N.v \begin{cases} > 0, & \text{Outside} \\ < 0, & \text{Inside} \\ = 0, & \text{Intersection} \end{cases}$$

Intersection point between segment lines (P_1,P_2) and (I_1,I_2) :

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0 \text{ and } \begin{vmatrix} x & y & 1 \\ x_{P1} & y_{P1} & 1 \\ x_{P2} & y_{P2} & 1 \end{vmatrix} = 0 \Rightarrow x = \frac{\begin{vmatrix} |A| & x_1 - x_2 \\ |B| & x_{P1} - x_{P2} \end{vmatrix}}{|C|}, y = \frac{\begin{vmatrix} |A| & y_1 - y_2 \\ |B| & y_{P1} - y_{P2} \end{vmatrix}}{|C|}$$

A $\qquad$ B

$$C = \begin{bmatrix} x_1 - x_2 & y_1 - y_2 \\ x_{P1} - x_{P2} & y_{P1} - y_{P2} \end{bmatrix}$$

Polygon Clipping-Sutherland-Hodgeman Algorithm

- This algorithm always generates single polygon in the output.
- This algorithm can operate for each clip-rectangle edge separately. Final result will be intersection of clip-edge results.

References

[1] H. Kiem, D.A. Duc, L.D. Duy, V.H. Quan, Cơ Sở Đồ Họa Máy Tính, NXB. Giáo Dục, 2005.

[2] D. Hearn, M.P. Baker, Computer Graphic: C Version in 2nd Ed., Prentice Hall, 1996.

[3] Foley, Van Dam, Feiner, Hughes, Computer Graphics - Principles and Practices 2nd Ed. In C, Addison Wesley, 1997.

[4] D.N.D.Tien, V.Q. Hoang, L. Phong, CG-Course Slide, HCM-University of Science.

Filled- Area Algorithms

Filled- Area Algorithms

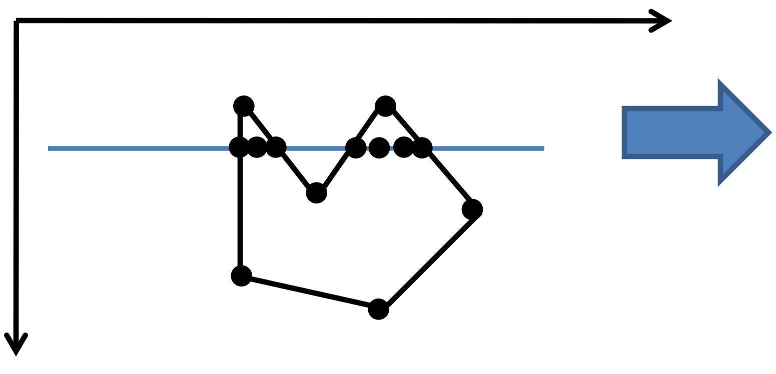
- Scan-line polygon fill algorithm
- Boundary fill algorithm

Color and Pattern Filling

Scan-line polygon fill algorithm

For each scan line,

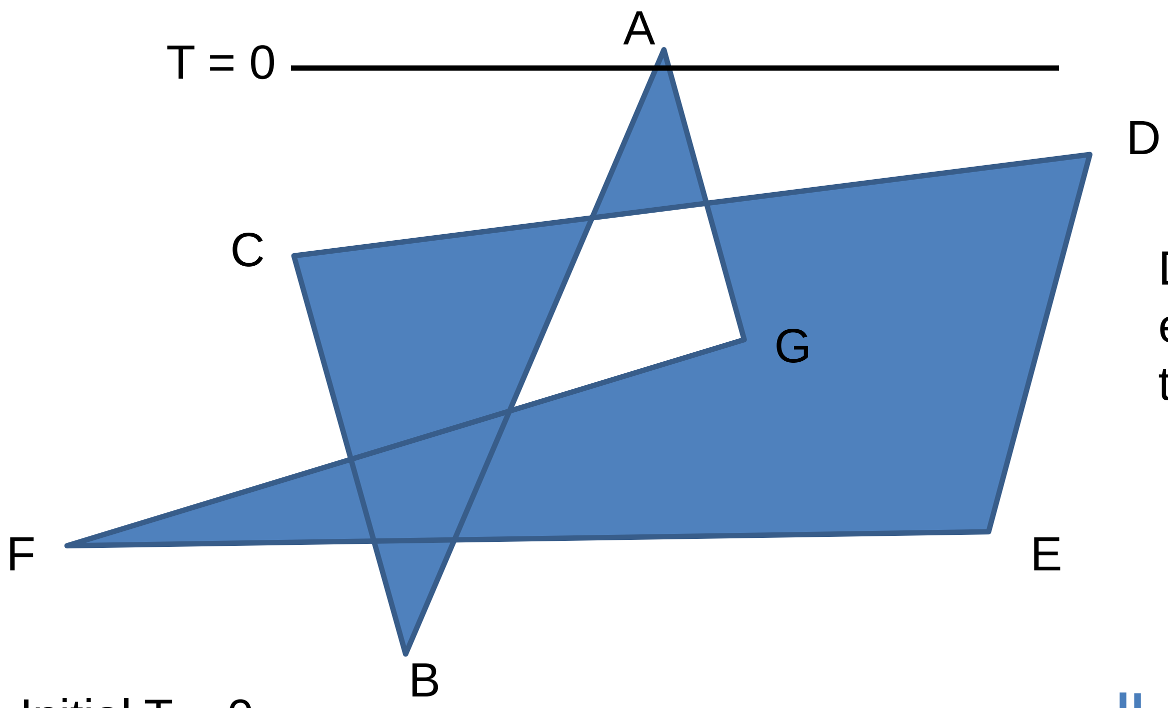
1. Find the intersections of the scan line with all edges of the polygon.
2. Sort the intersections by increasing x coordinate.
3. Fill in all pixels between pairs of intersections that are inside the polygon.



Based on the intersections, how can we find interior pixels of polygon ?

Scan-line polygon fill algorithm

- Odd-even (odd-parity) rule:



Draw from odd to even and from left to right.

- Initial T = 0
- Cross one edge T = T +1
- Cross two edges T = T
- Cross one vertex T = T if vertex is a valley or peak else T = T + 1

Is there any counterexample ?

Scan-line polygon fill algorithm

- Let y_{min} and y_{max} denote the maximum and minimum values of vertical coordinate.

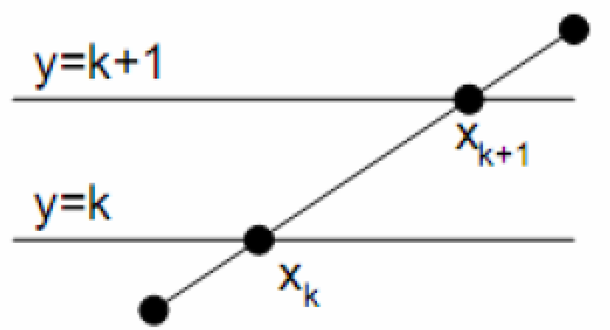
$$y_{\min} = \min\left\{ y_i, (x_i, y_i) \in \text{Polygon} \right\}$$
$$y_{\max} = \max\left\{ y_i, (x_i, y_i) \in \text{Polygon} \right\}$$

- For each $y=y_k$, $y_{min} \leq y_k \leq y_{max}$, calculate intersections and sort by increasing x coordinate.
- Using odd-even rule, calculate T value for each intersection.
- Draw from odd-T intersection until we meet even-T intersection or go to next case if there is not any even-T intersection follow odd-T point.

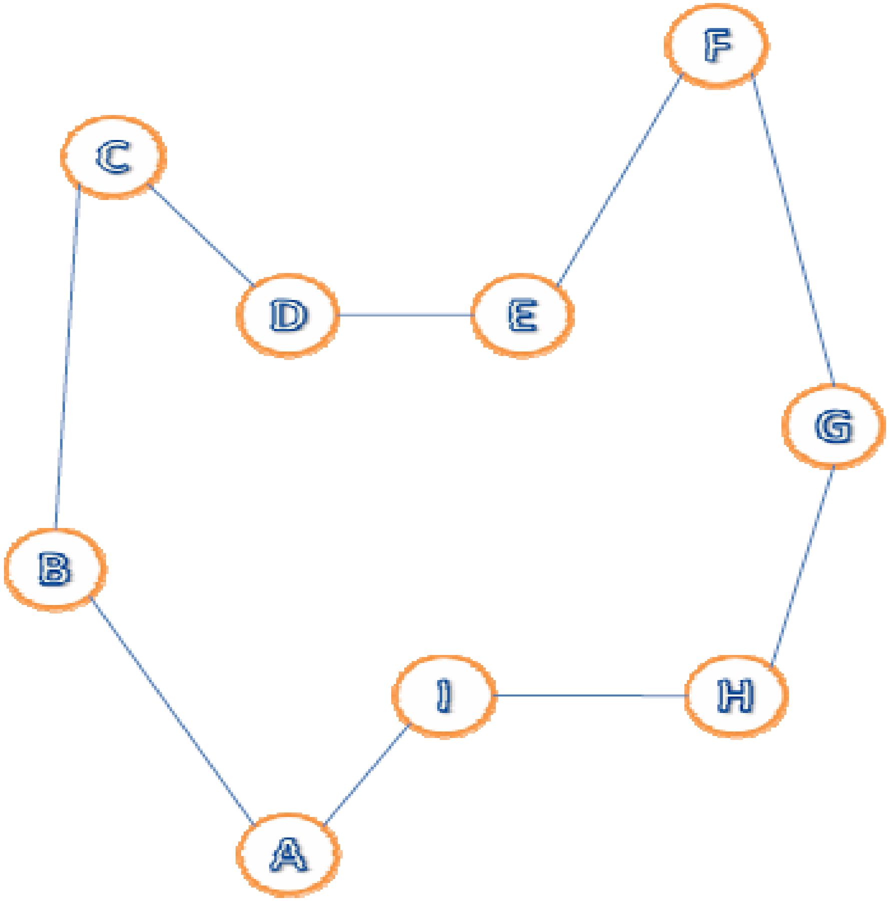
Scan-line polygon fill algorithm

- Speed up finding intersection points by using previous results

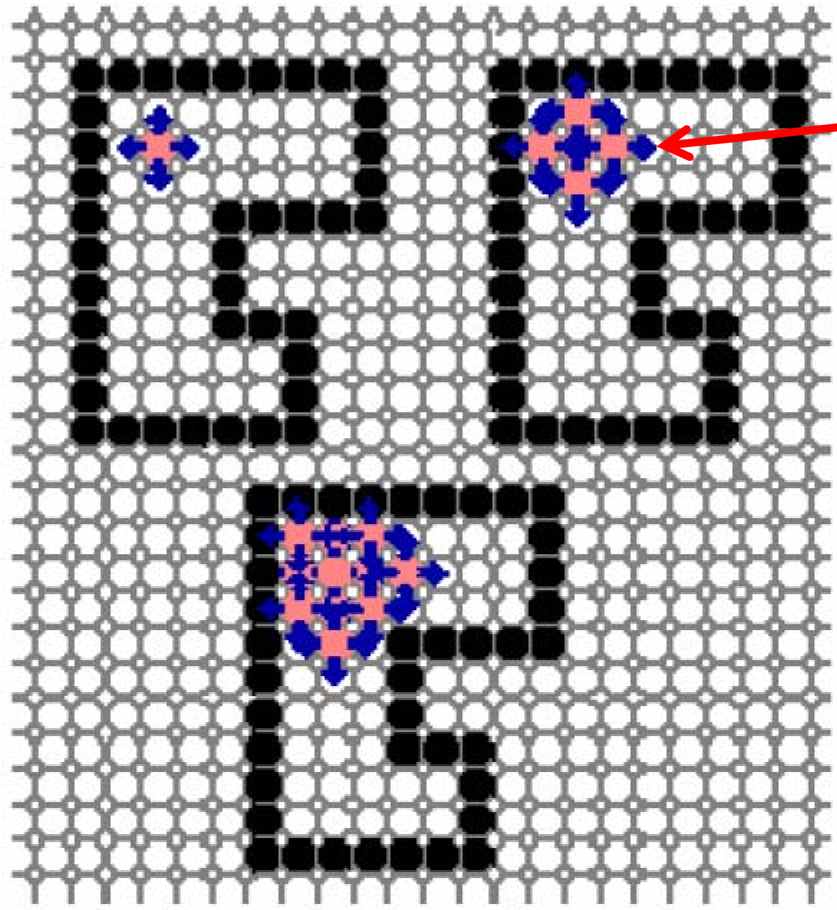
$$x_{k+1} - x_k = \frac{1}{m}\left((k+1) - k\right) = \frac{1}{m} \text{ hay } x_{k+1} = x_k + \frac{1}{m}.$$



Example



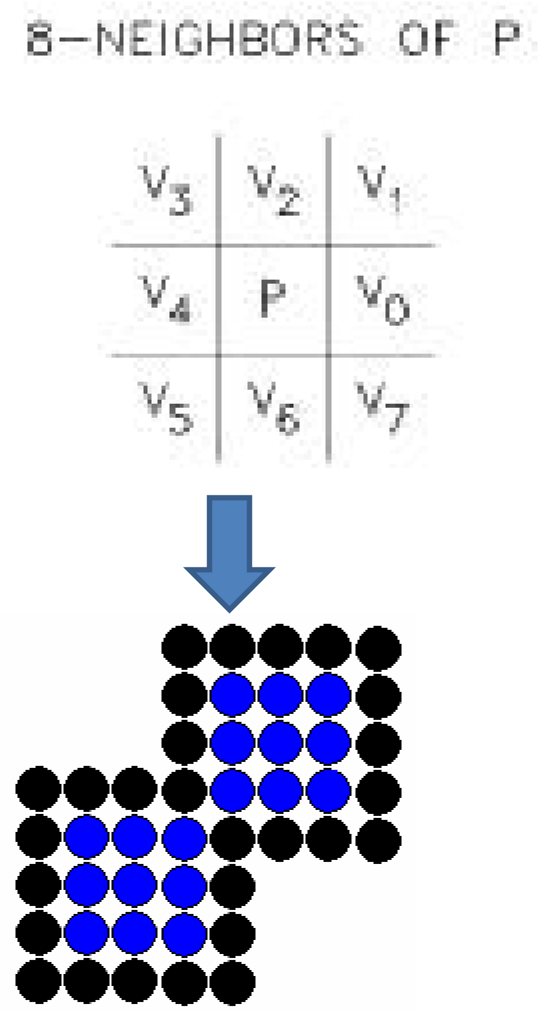
Boundary-Fill Algorithm

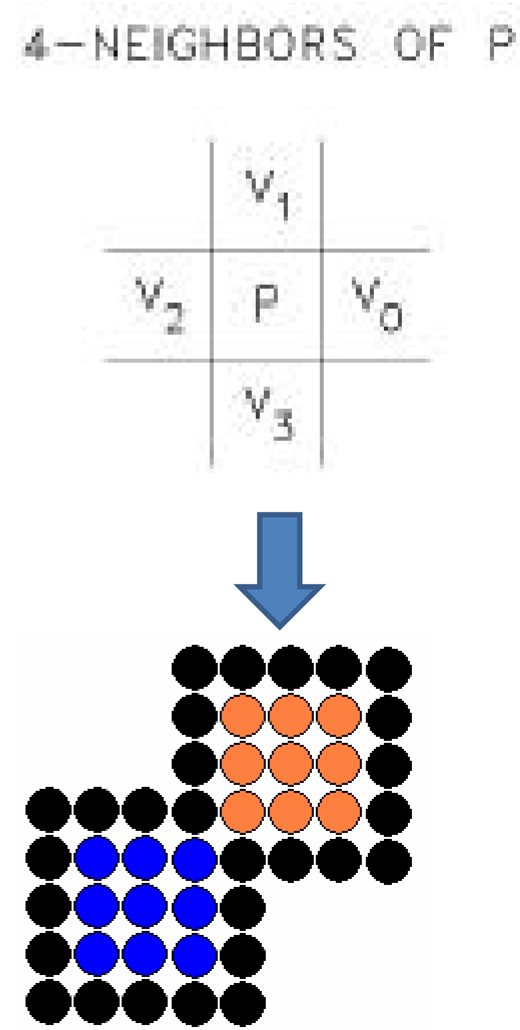


1. Start at one point inside polygon.
2. Paint the interior outward toward the boundary by using connected part of the starting point.

Boundary-Fill Algorithm

- Connected part:





Boundary-Fill Algorithm

- Recursive algorithm:

```
void BoundaryFill(int x, int y, int FillColor, int BoundaryColor)
{
        int CurrenColor;
        CurrentColor = getpixel(x,y);
        if((CurrentColor!=BoundaryColor)&&CurrentColor!=
FillColor))
        {
                putpixel(x,y,FillColor);
                BoundaryFill(x-1, y, FillColor, BoundaryColor);
                BoundaryFill(x, y+1, FillColor, BoundaryColor);
                BoundaryFill(x+1, y, FillColor, BoundaryColor);
                BoundaryFill(x, y-1, FillColor, BoundaryColor);
        }
} // Boundary Fill
```

Flood-Fill Algorithm

- Find the seeds which satisfy conditions

  - Position inside the boundary of polygon.

  - Unpainted points.

  - Its left neighbor is a boundary point or one painted points.

  - Or it is a neighbor of another seeds.

Flood-Fill Algorithm

- Step 1: Generate an empty stack.
- Step 2: Select and push one seed in stack from the top (or bottom) of the object.
- Step 3:

  + Pop one seed out of stack and paint to all left and right points corresponding to its row until we meet the boundary or the painted point.

  + Search another seed in the current row or in the next row. If there is one seed, push it in stack.
- Step 4: Go to Step 3 until stack is empty.

References

[1] H. Kiem, D.A. Duc, L.D. Duy, V.H. Quan, Cơ Sở Đồ Họa Máy Tính, NXB. Giáo Dục, 2005.

[2] D. Hearn, M.P. Baker, Computer Graphic: C Version in 2nd Ed., Prentice Hall, 1996.

[3] Foley, Van Dam, Feiner, Hughes, Computer Graphics - Principles and Practices 2nd Ed. In C, Addison Wesley, 1997.

[4] D.N.D.Tien, V.Q. Hoang, L. Phong, CG-Course Slide, HCM-University of Science.