

# COMP 371 -- Winter 2012

# Computer Graphics

2D Graphics Algorithms  
Scan Conversion  
Antialiasing  
Clipping

# Recap - 1

- We digitally model the world of interest using geometric primitives.
- For rendering this world, we must compute the part of the world that is visible within a pixel. And then compute the pixel colour such that it visually represents that part of the world as best as it can.
- Edges (geometrically approximated by straight lines) dominate visual form perception by humans.
- How do we render edges in our digitally created world?

# Property of Lines

- Point classification is a very fundamental operation in computer graphics. It is defined as follows:
  - *Given a point  $P$  and a geometric entity  $E$ , classify the point as being outside, on or inside the entity.*
- Example:
  - $P = (x_0, y_0)$
  - Let  $E$  be a line with equation:  $ax+by+c=0$
  - Let  $v = ax_0+by_0+c$ .
  - Then  $P$  is inside if  $v<0$ , on if  $v=0$  and outside if  $v>0$

# Raster Graphics

- Display devices have evolved since the mid sixties
- Vector displays were the first to appear and were commonly used until the mid-eighties
- Raster displays based on the television technology were first developed in the early seventies
- These became popular mainly because
  - they were cheap due to the simple hardware, unlike vector displays
  - supported solid shaded or patterned regions
  - can produce highly realistic images of digital world models

# Rasterization

- The raster display is a matrix of picture elements also called *pixels*. Each pixel has a color value assigned.
- A frame buffer stores the values for each pixel.
- The task of displaying a world modeled using primitives like lines, polygons, filled/patterned areas, etc. can be carried out in two steps
  - determine the pixels through which the primitive is visible, a process called *Rasterization or scan conversion*
  - determine the color value to be assigned to each such pixel.

# Raster Graphics Packages

- The efficiency of these steps forms the main criteria to determine the performance of a display
- The raster graphics package is typically a collection of efficient algorithms for *scan converting* (rasterization) of the display primitives
- High performance graphics workstations have most of these algorithms implemented in hardware

# Why Study these Algorithms?

- Some of these algorithms are very good examples of clever algorithmic optimization done to dramatically improve performance using minimal hardware facilities.

## Story:

Ptolemy, the young reigning king of Egypt once asked Euclid:

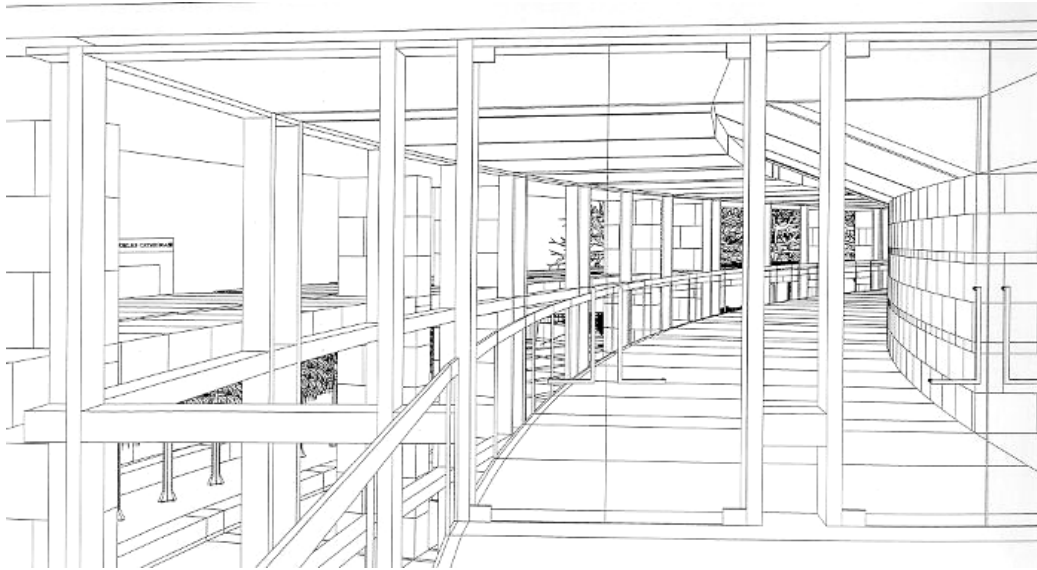
*Is there no quicker way to master geometry than by studying your Elements ?*

Euclid replied:

*Your Highness. There is no royal road to geometry.*

# Scan Converting a Line Segment

- The line is a powerful element used since the days of Euclid to model the edges in the world.



- Given a line segment defined by its endpoints determine the pixels and colour which best model the line segment.

# Scan converting lines

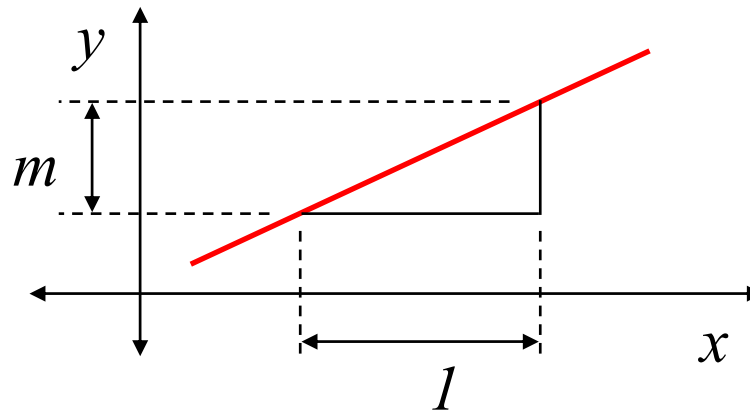
- Requirements
  - the chosen pixels should lie as close to the ideal line as possible
  - the sequence of pixels should be as straight as possible. All lines should appear to be of constant brightness independent of their length and orientation
  - should start and end accurately
  - should be drawn as rapidly as possible
  - should be possible to draw lines with different width and line styles

# Equation of a Line

- Equation of a line is  $y - m \cdot x + c = 0$
- For a line segment joining points  $P(x_1, y_1)$  and  $P(x_2, y_2)$

$$\text{slope } m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$$

- Slope  $m$  means that for every unit increment in  $X$  the increment in  $Y$  is  $m$  units



# Digital Differential Analyzer (DDA)

- We consider the line in the first octant. Other cases can be easily derived.
- Uses differential equation of the line

$$y_i = m \cdot x_i + c$$

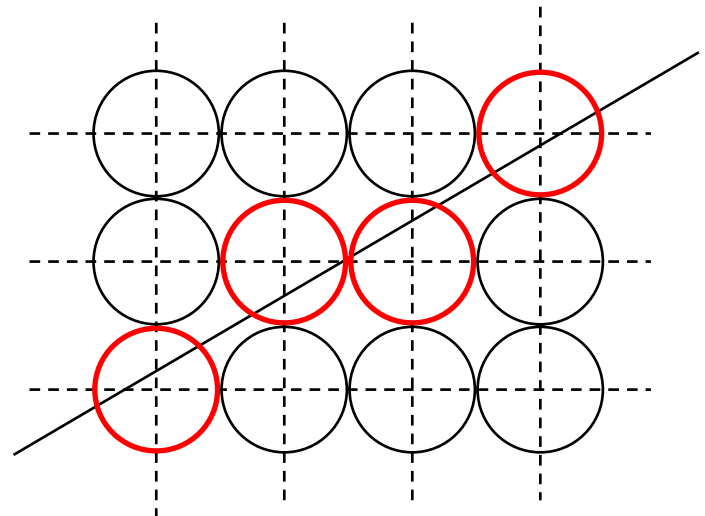
$$\text{where, } m = \frac{y_2 - y_1}{x_2 - x_1}$$

- Incrementing X-coordinate by 1

$$x_i = x_{i\_prev} + 1$$

$$y_i = y_{i\_prev} + m$$

- Illuminate the pixel  $[x_i, \text{round}(y_i)]$

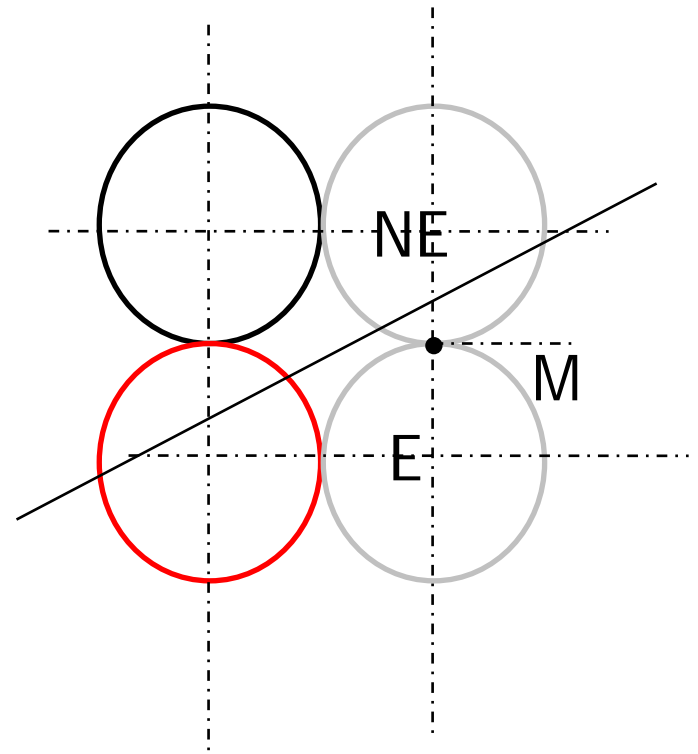


# Digital Differential Analyzer

- Digital Differential Analyzer algorithm more popularly known as *DDA*
- This is an *Incremental algorithm* i.e. at each step it makes incremental calculations based on the calculations done during the preceding step
- The algorithm uses floating point operations, which are very cleverly avoided in an algorithm first proposed by J. Bresenham of IBM. The algorithm is well known as *Bresenham's Line Drawing Algorithm*.
- We will discuss a slight variation – Midpoint Line Drawing Algorithm

# Midpoint Line Drawing

- The same incremental method for scan converting lines can be derived using an integer formulation. In this the mid-point between the East (E) and NorthEast (NE) pixels is checked to see on which side of the line it lies. For this instead of  $y = mx + c$ , the line equation of the form
- $F(x,y) = ax + by + c$  is used.
- **Principle:** If  $F(\text{mid-point})$  is above ( $\leq 0$ ) the line then E is chosen, otherwise NE is chosen.



# Midpoint Line Algorithm

$$F(M) = F(x_p, y_p) = d$$

$$= a(x_p + 1) + b(y_p + 1/2) + c$$

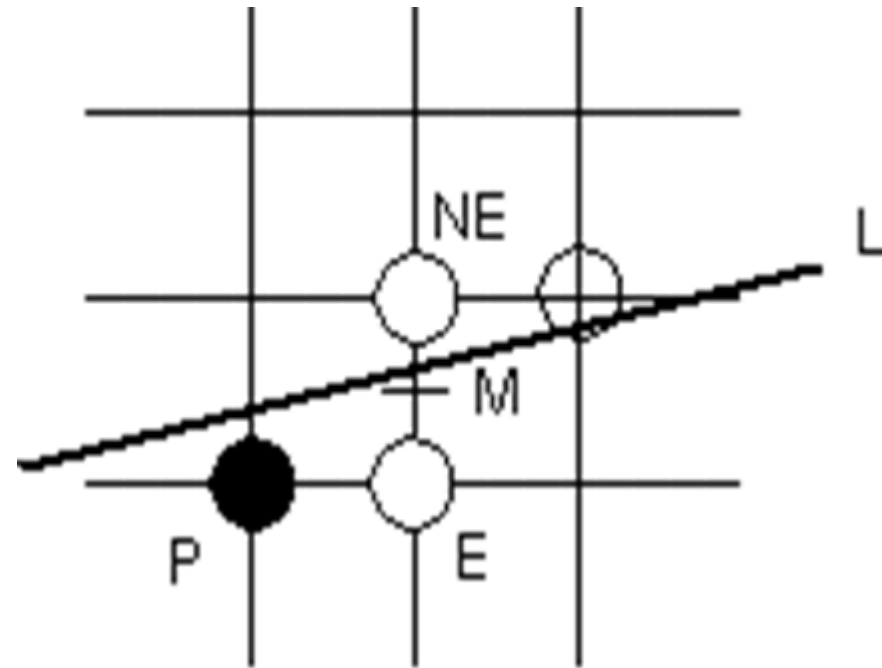
is the decision variable.

if  $d > 0$  then choose NE else if  $d \leq 0$  then choose E.

For an incremental algorithm, we must compute  $d$  incrementally.

For that, let us see what happens to  $M$  and  $d$  for the next grid line.

We have two cases – old choice was E or NE



# Midpoint Line Algorithm - contd

If the old choice is E, then

$$d_{\text{new}} = a(x_p + 2) + b(y_p + 1/2) + c$$

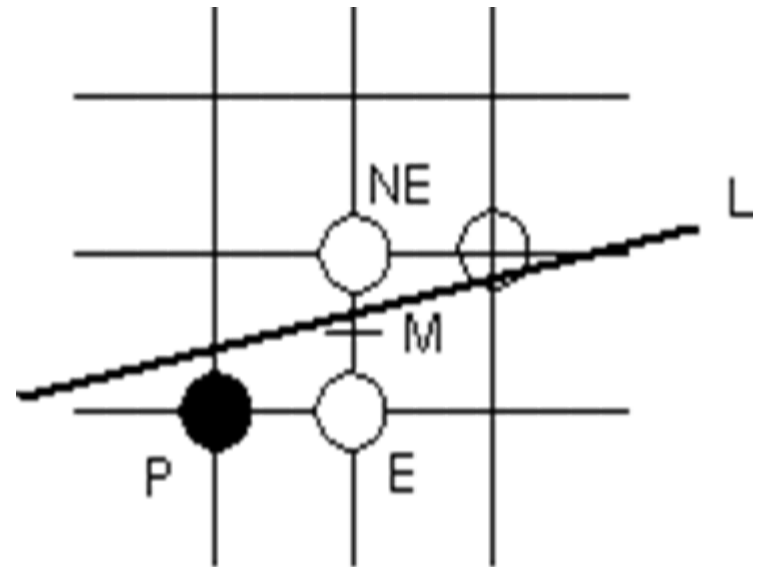
$$\text{But } d_{\text{old}} = a(x_p + 1) + b(y_p + 1/2) + c$$

$$\text{Hence } d_{\text{new}} = d_{\text{old}} + a.$$

If the old choice is NE, then

$$d_{\text{new}} = a(x_p + 2) + b(y_p + 3/2) + c$$

$$\text{Now } d_{\text{new}} = d_{\text{old}} + a + b.$$



## Midpoint Line Algorithm - contd

Consider the line segment from  $(x_1, y_1)$  to  $(x_2, y_2)$ .

$(y_2 - y_1)x - (x_2 - x_1)y + c = 0$  is the equation.

$a = dy = (y_2 - y_1)$ ,  $b = -dx = -(x_2 - x_1)$

And  $a + b = dy - dx$

What should be  $d$  be to start with?

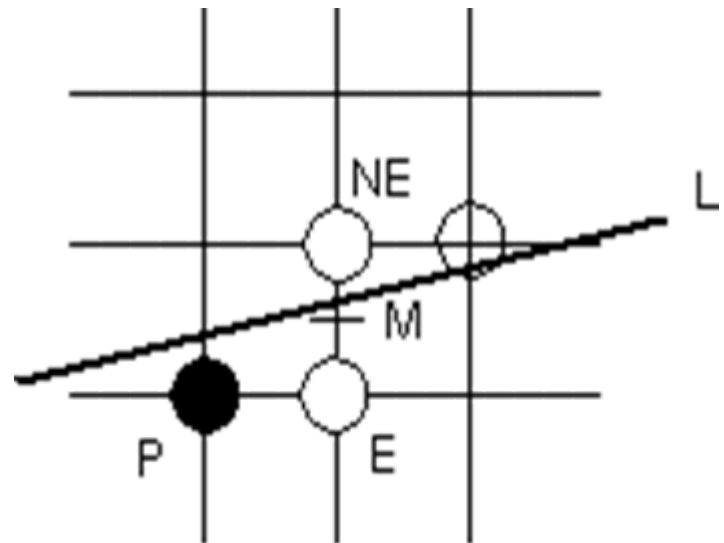
The first midpoint  $M_1 = (x_1 + 1, y_1 + 1/2)$

$$\begin{aligned} F(M_1) &= d_1 = ax_1 + by_1 + c + a + b/2 \\ &= F(x_1, y_1) + a + b/2 \end{aligned}$$

$x_1, y_1$  is on the line, so  $F(x_1, y_1) = 0$

Thus  $d_1 = a + b/2$

In order to avoid division by 2, we choose to make our decision using  $2d_1 = 2a + b$ , which does not change sign of  $d$ .



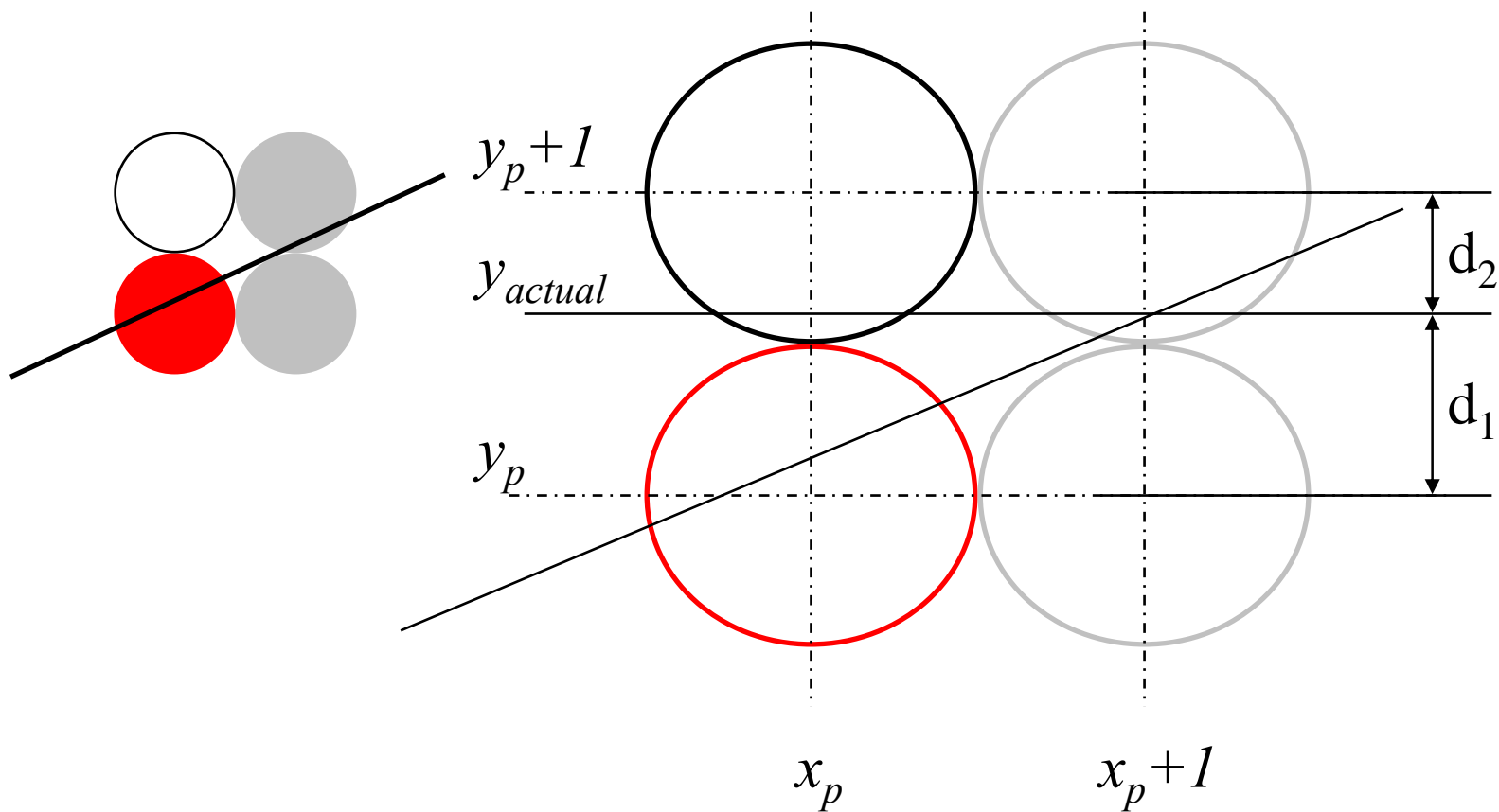
# Midpoint Line Algorithm

- Input line end points  $(x_1, y_1), (x_2, y_2)$
- Set  $x = x_1$  and  $y = y_1$  and  $\text{SetLineColour}(x, y)$
- Calculate  $dX = x_2 - x_1$  and  $dY = y_2 - y_1$
- Calculate  $\text{incrNE} = 2 * (dY - dX)$  and  $\text{incrE} = 2 * dY$
- Calculate  $d = 2 * dY - dX$
- While  $(x < x_2)$ 
  - {  $x = x + 1;$
  - If  $d > 0$  then  $y = y + 1$  and  $d = d + \text{incrNE}$  else  $d = d + \text{incrE};$
  - $\text{SetLineColour}(x, y)$
  - }

# Advantages of Incremental Midpoint Line Algorithm

- It is an incremental algorithm
- It uses only integer arithmetic
- Provides the best fit approximation to the actual line

# Bresenham Line Drawing

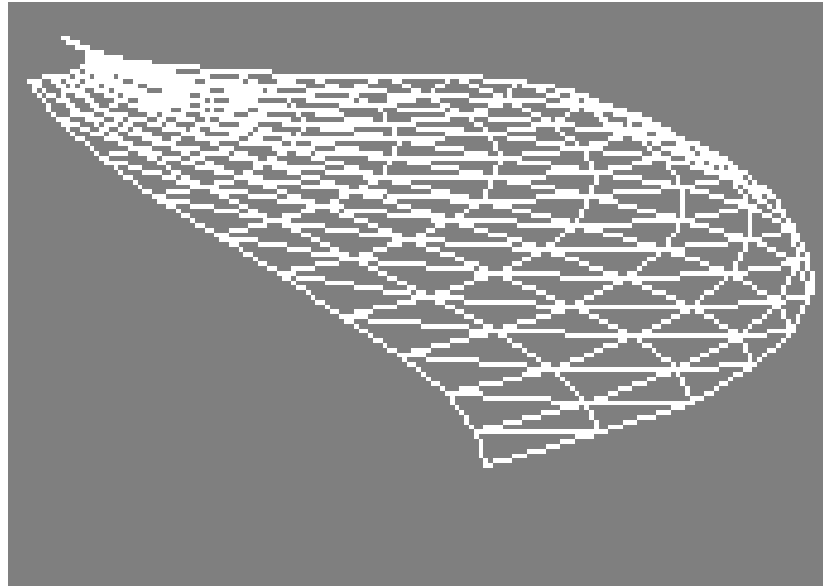


# Some Additional Issues

- How do we handle line segments in other octants?
- Drawing should be endpoint order independent:  
    P0 - P1 should be identical to P1-P0
- If line segment goes outside the raster window, it has to be clipped; then start is no longer at the centre of any pixel.  $d_0$  has to be suitably computed.
- Variation of intensity as function of slope – (cannot be avoided without antialiasing)
- Connected line segments. Common end points may be drawn multiple times. Could give rise to problems sometimes.

# Polygon Surfaces

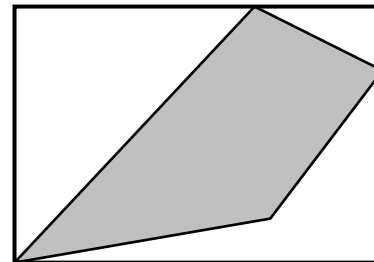
- Basic form of representation in most applications – all real-time displays.
- Easy to process, fast to process.
- Some applications may allow other descriptions, e.g., splines, but reduce all objects to polygons for processing.



Bézier surface as a triangle mesh

# Polygon filling

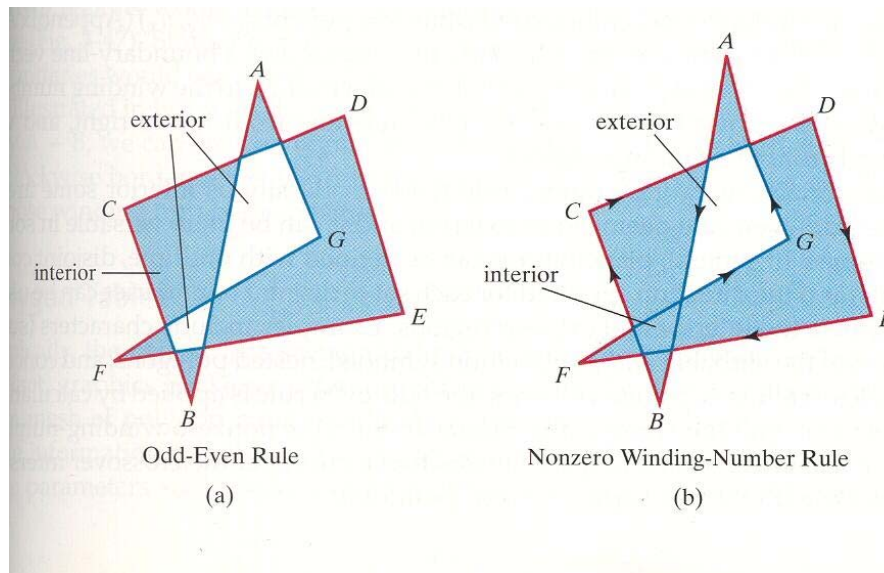
- Simplest method to fill a polygonal area is to test every pixel in the raster to see if it lies inside the polygon.
- There are two methods to make an inside check
  - even-odd test
  - non-zero winding number test
- Bounding boxes can be used to improve performance



Bounding box

# Polygon filling

- Simplest method to fill a polygonal area is to test every pixel in the raster to see if it lies inside the polygon.
- There are two methods to make an inside check
  - even-odd test – # of line-segment crossings by ray from point P (odd-interior, even-exterior)
  - winding number test – add 1 to winding number, if line segment crosses the ray from right to left, otherwise subtract 1 (nonzero-interior, zero-exterior)
- Bounding boxes can be used to improve performance

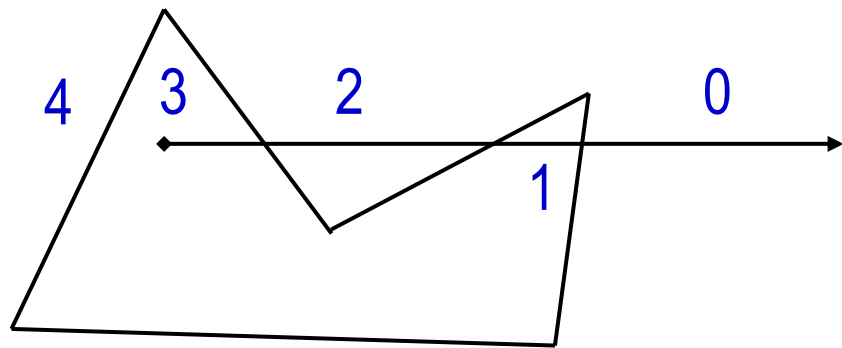


# Jordan Curve Theorem.

Testing for inside/outside of a polygon.

Two definitions of inside :

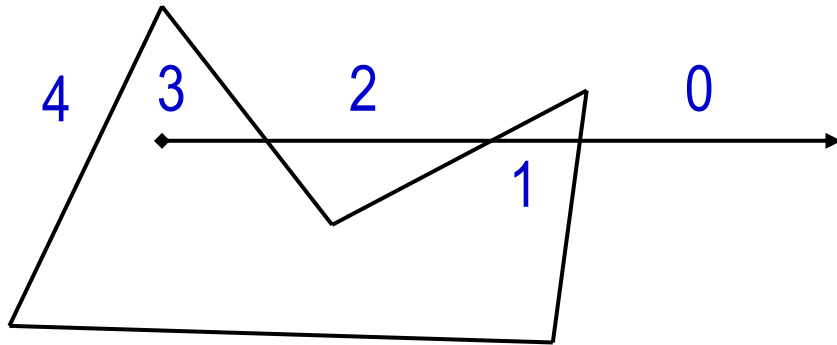
- Even-Odd parity
- Non-zero Winding number



Even no. crossings : Outside polygon

Odd no. crossings : Inside polygon.

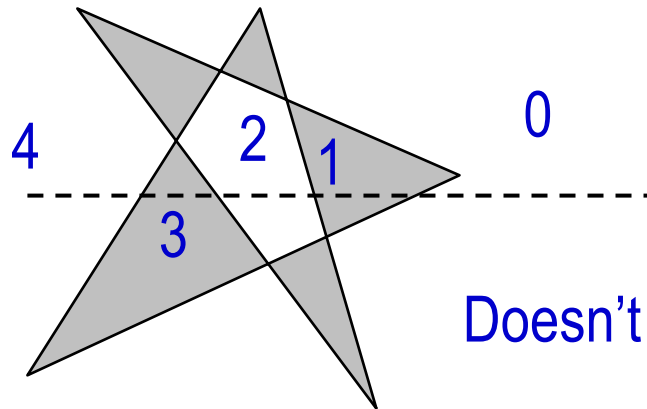
# Even-Odd Parity Test



Even-Odd parity

Even no. crossings : Outside polygon

Odd no. crossings : Inside polygon.

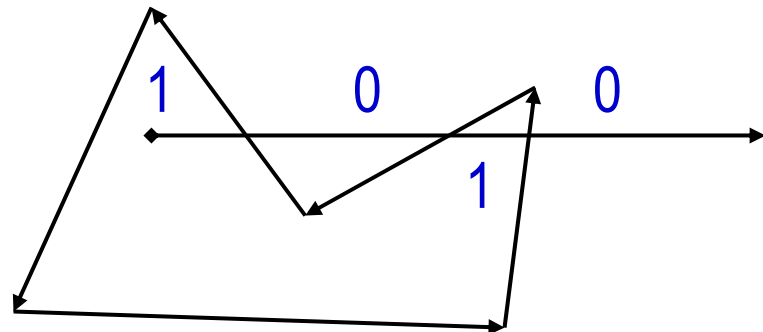
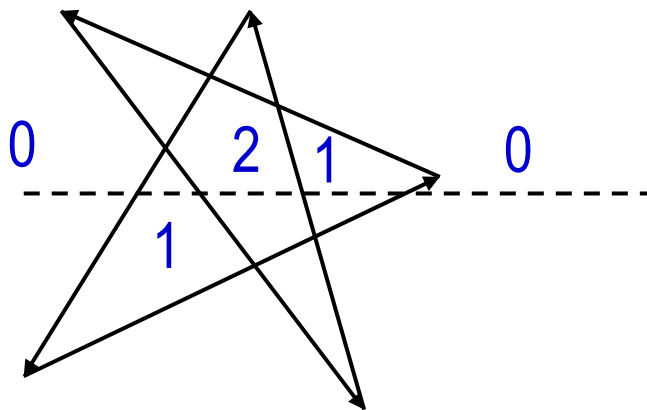


Doesn't work for self-intersecting polygons

# Non-zero Winding Number Test.

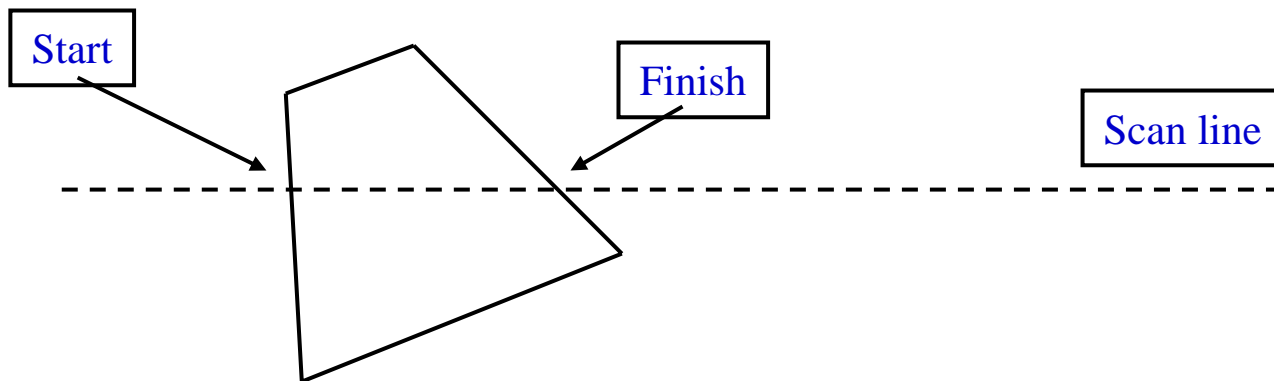
## Non-zero Winding Number

- Use direction of line as well as crossing
- Set a value,  $e = 0$
- For right-left crossings,  $e++$ , for left-right  $e--$
- For inside,  $e \neq 0$



# Simple scan conversion

- May need to fill the polygon with colour(s).
- Incorporate in a scan conversion algorithm.

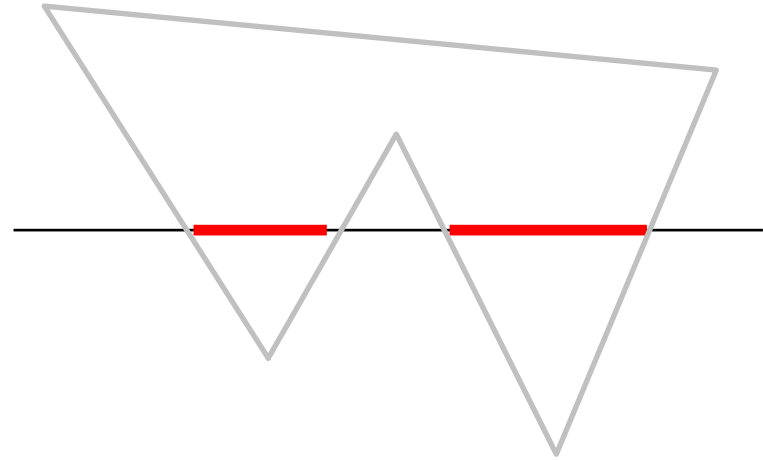


# Scan Line Methods

- Makes use of the *coherence* properties
  - spatial coherence : Except at the boundary edges, adjacent pixels are likely to have the same characteristics
  - span coherence : Pixels in a scan line will be set to same values for solid shaded primitives
  - scan line coherence : Pixels in the adjacent scan lines are likely to have the same characteristics
- Uses intersections between area boundaries and scan lines to identify pixels that are inside the area

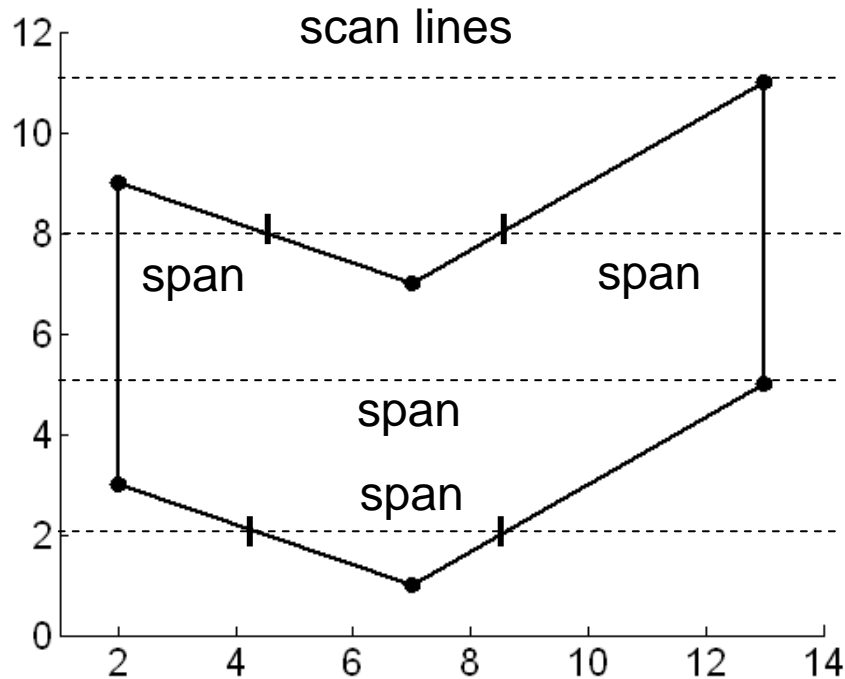
# Scan Line Method

- Proceeding from left to right the intersections are paired and intervening pixels are set to the specified intensity



- Algorithm
  - Find the intersections of the scan line with all the edges in the polygon
  - Sort the intersections by increasing X-coordinates
  - Fill the pixels between pair of intersections

# Illustration of the Ideas



A **span** is the collection of adjacent pixels on a single scan line which lie inside the primitive.

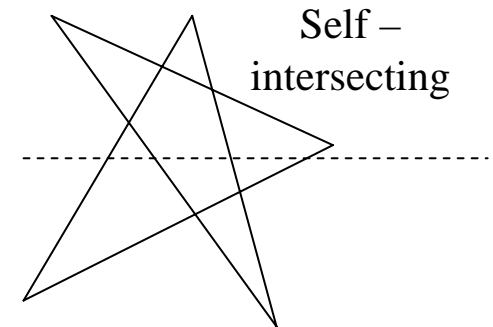
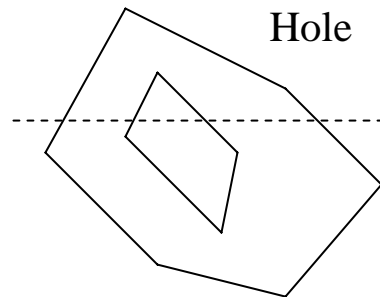
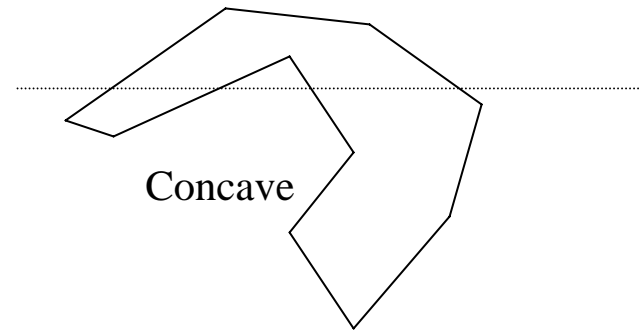
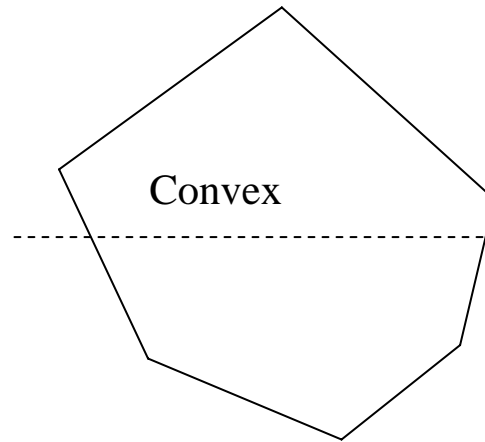
**Coherence** literally means *to be logically consistent or connected*. **Spatial coherence** means that primitives don't change an awful lot if at all from pixel to pixel within a scan line or from scan line to scan line. This allows us to optimise our algorithms.

**Edge coherence** means that most of the edges that intersect scan line  $i$  also intersect scan line  $i+1$ .

# Types of polygons.

## Types

- Triangles
- Trapezoids
- Quadrilaterals
- Convex
- Concave
- Self-intersecting
- Multiple loops
- Holes

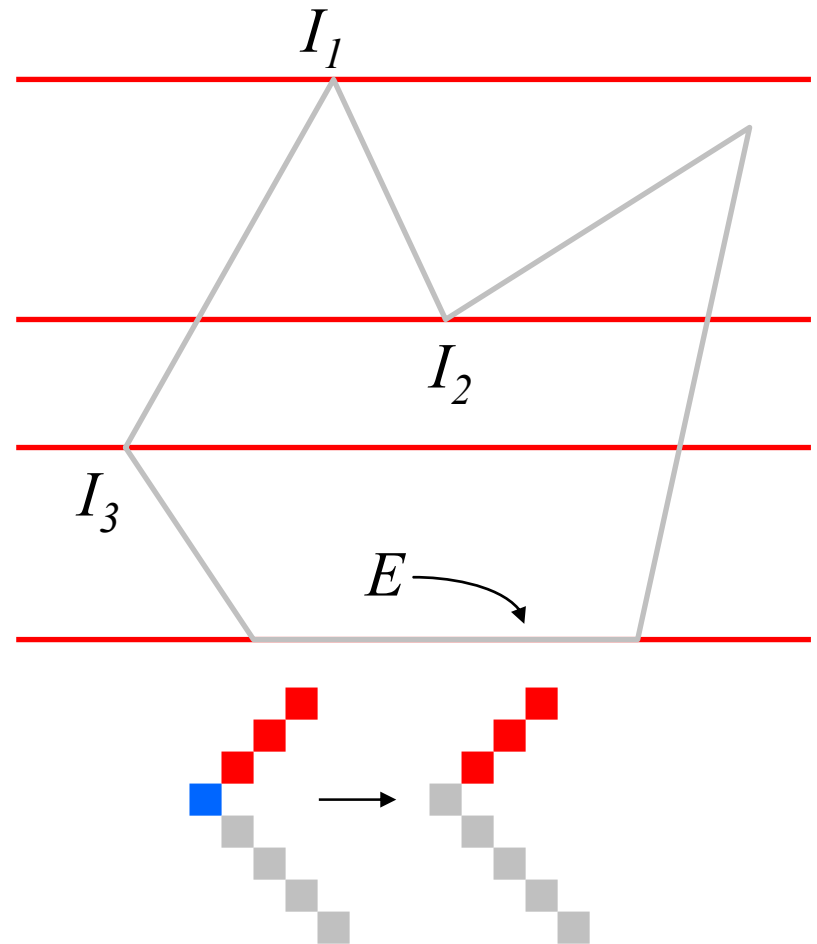


Two approaches :

- Generalise scan conversion
- Split into triangles.

# Special cases for Scan Line Method

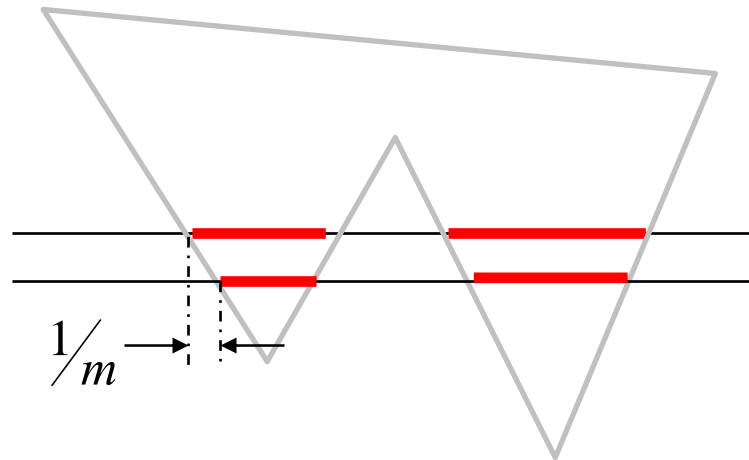
- Overall topology should be considered for intersection at the vertices
- Intersections like  $I_1$  and  $I_2$  should be considered as two intersections
- Intersections like  $I_3$  should be considered as one intersection
- Horizontal edges like  $E$  need not be considered



# Efficiency Issues in Scan Line Method

- Intersections could be found using edge coherence
  - the X-intersection value  $x_{i+1}$  of the lower scan line can be computed from the X-intersection value  $x_i$  of the preceding scanline as

$$x_{i+1} = x_i + 1/m$$



(i.e.,  $\Delta x = 1/m$  or  $x_{i+1} = x_i + 1/m$  since  $\Delta y = 1$ ).

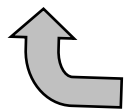
- List of active edges could be maintained to increase efficiency
- Efficiency could be further improved if polygons are convex, much better if they are only triangles

# Triangles are Always Convex

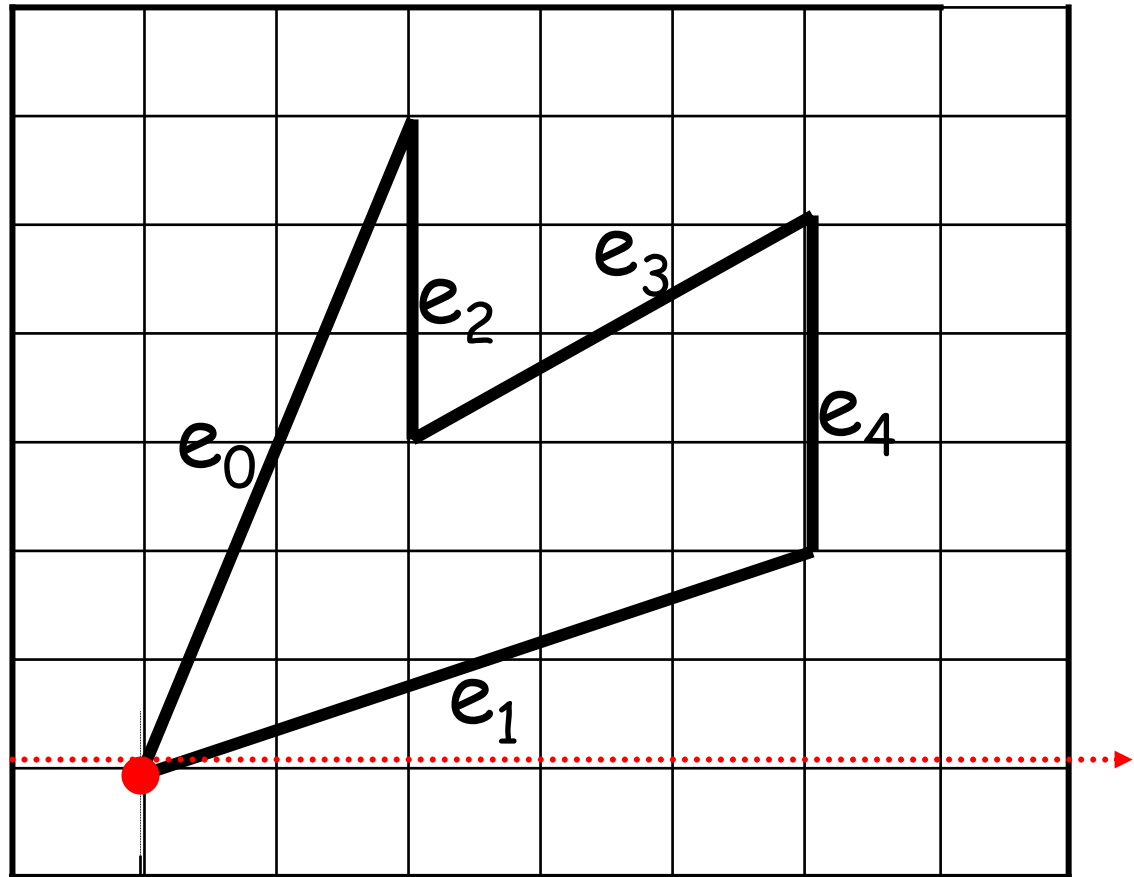
- Mathematically very simple – involving simple linear equations.
- Three points guaranteed co-planer.
- Any polygon can be decomposed into triangles.
- Triangles can approximate arbitrary shapes.
- For any orientation on screen, a scan line will intersect only a single segment (scan).
- Hardware acceleration usually available.

# Implementation: Sorted Edge Table

edge	$y_{\text{bottom}}$	$y_{\text{top}}$	$x_{\text{int}}$	$1/m$
$e_0$	1	7	1	$1/3$
$e_1$	1	3	1	$5/2$
$e_4$	3	6	6	0
$e_2$	4	7	3	0
$e_3$	4	6	3	$3/2$



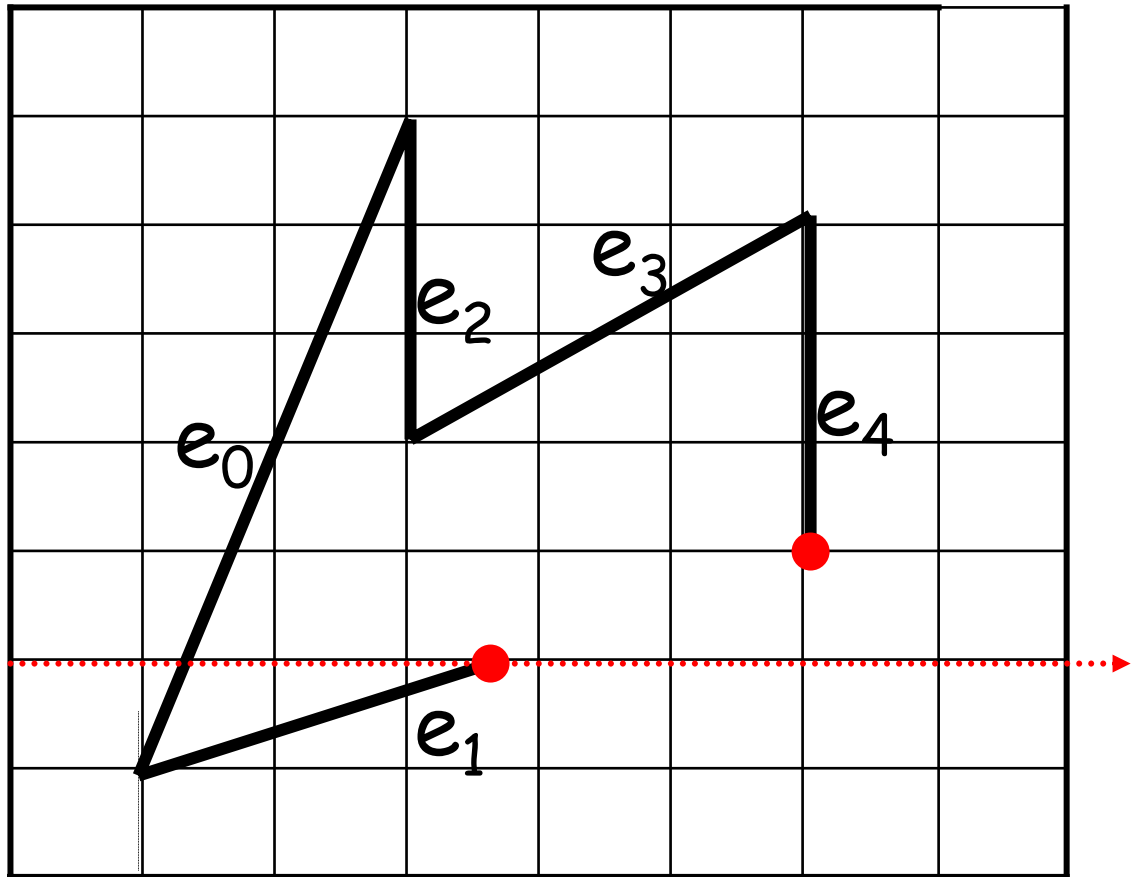
Sorted by  
smallest  $y$  value



1  $x_{\text{int}}$  --  $x$ -intercept value at lower  
vertex for an edge

# Implementation: Sorted Edge Table

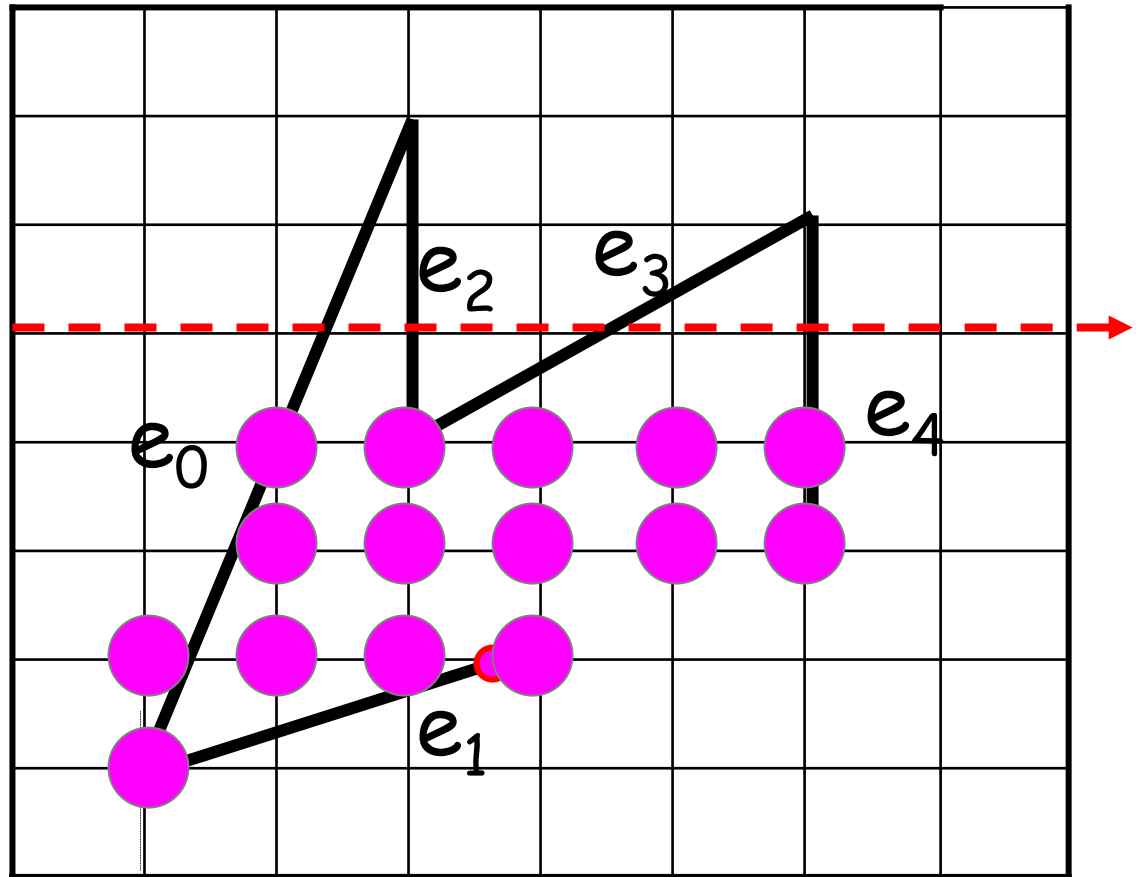
edge	$y_{\text{bottom}}$	$y_{\text{top}}$	$x_{\text{int}}$	$1/m$
$e_0$	1	7	1	$1/3$
$e_1$	1	2	1	$5/2$
$e_4$	3	6	6	0
$e_2$	4	7	3	0
$e_3$	4	6	3	$3/2$



$e_1$  is shortened by one unit in the  $y$  direction.

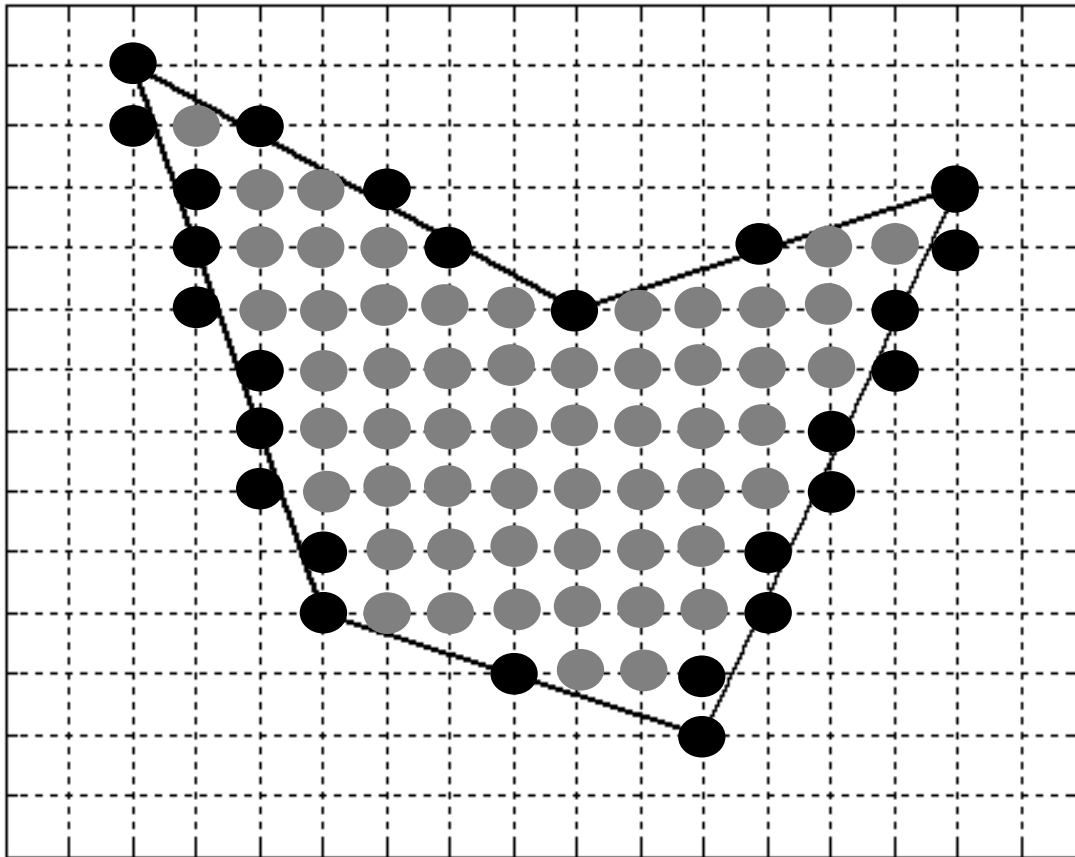
# Implementation: Active Edge List

edge	$x_{\text{int}}$	...
$e_0$	1	...
$e_2$	3	...
$e_3$	3	...
$e_4$	6	...



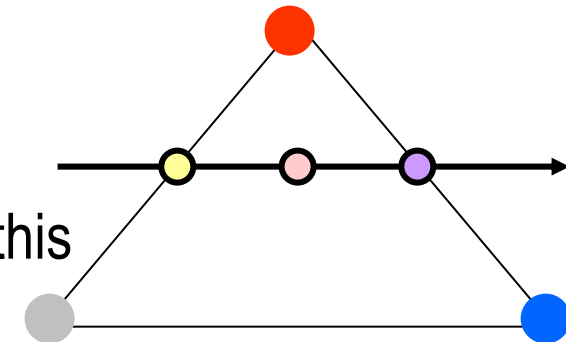
# Advantages of Scan Line method

- The algorithm is efficient -- each pixel is visited only once



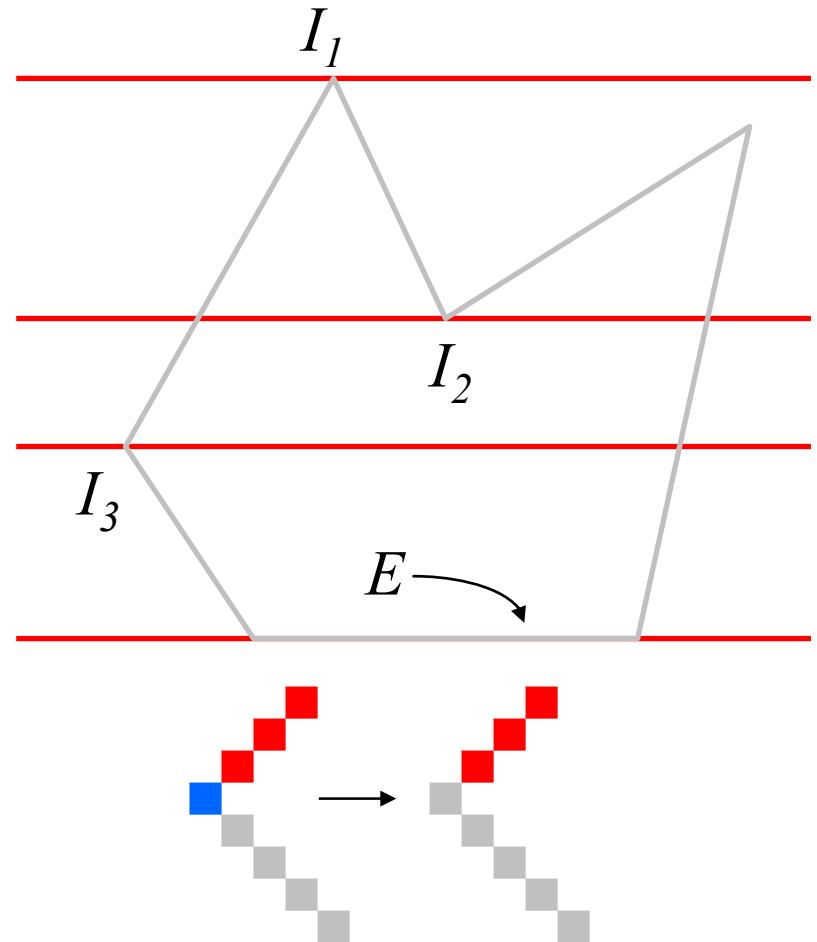
- : intersection points between scan lines and edges (span extrema).
- : interior pixels

- Shading algorithms could be easily integrated with this method to obtain shaded area



# Special cases for Scan Line Method

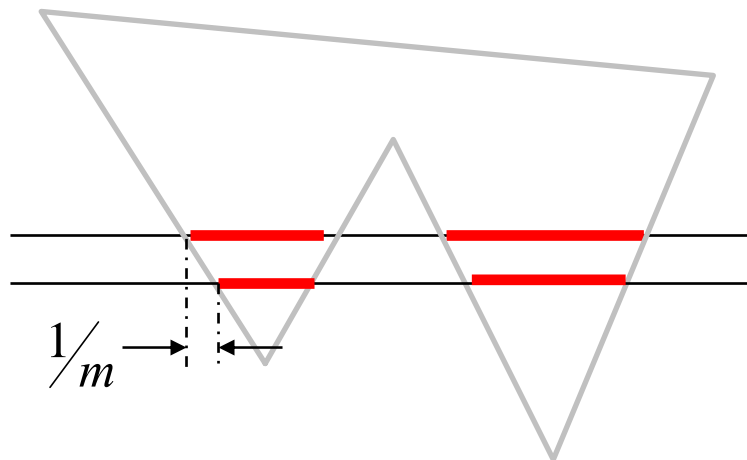
- Overall topology should be considered for intersection at the vertices
- Intersections like  $I_1$  and  $I_2$  should be considered as two intersections
- Intersections like  $I_3$  should be considered as one intersection
- Horizontal edges like  $E$  need not be considered



# Efficiency Issues in Scan Line Method

- Intersections could be found using edge coherence
  - the X-intersection value  $x_{i+1}$  of the lower scan line can be computed from the X-intersection value  $x_i$  of the preceding scan line as

$$x_{i+1} = x_i + \frac{1}{m}$$



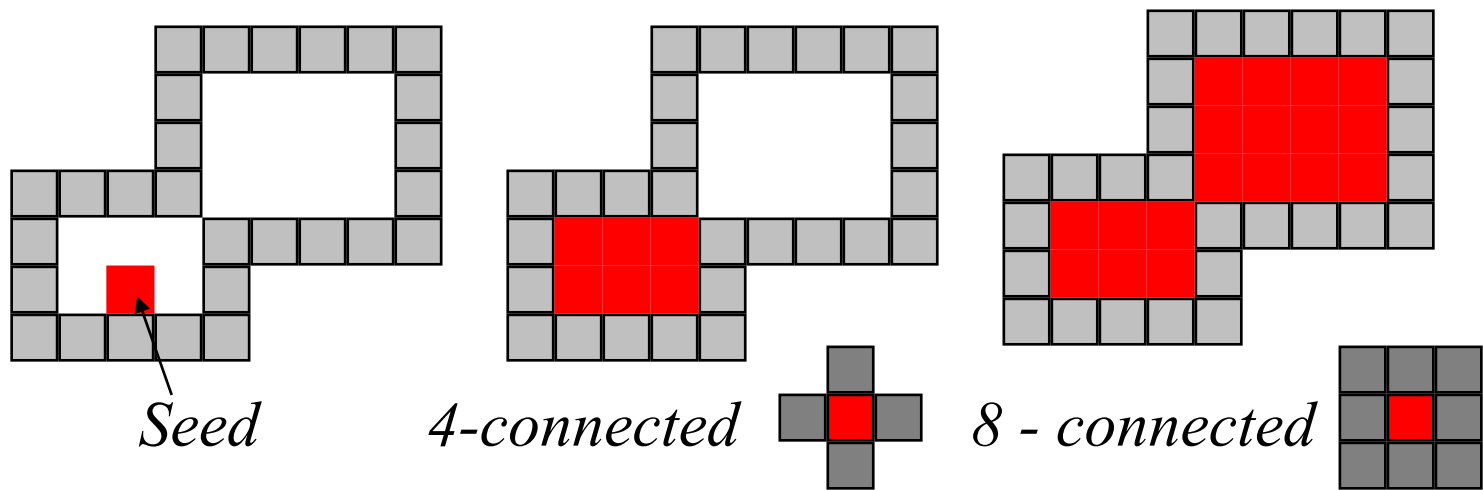
- List of active edges could be maintained to increase efficiency
- Efficiency could be further improved if polygons are convex, much better if they are only triangles

# Advantages of Scan Line method

- The algorithm is efficient
- Each pixel is visited only once
- Shading algorithms could be easily integrated with this method to obtain shaded area

# Seed Fill Algorithms

- Assumes that at least one pixel interior to the polygon is known.
- It is a recursive algorithm
- Useful in interactive paint packages



# Anti-Aliasing

# Aliasing

- Aliasing is caused due to the discrete nature of the display device
- Rasterizing primitives is like sampling a continuous signal i.e. representing a signal by a finite set of values (point sampling)
- Information is lost if the rate of sampling is not sufficient. This sampling error is called *aliasing*.
- Effects of aliasing are
  - Jagged edges
  - Incorrectly rendered fine details
  - Small objects might miss



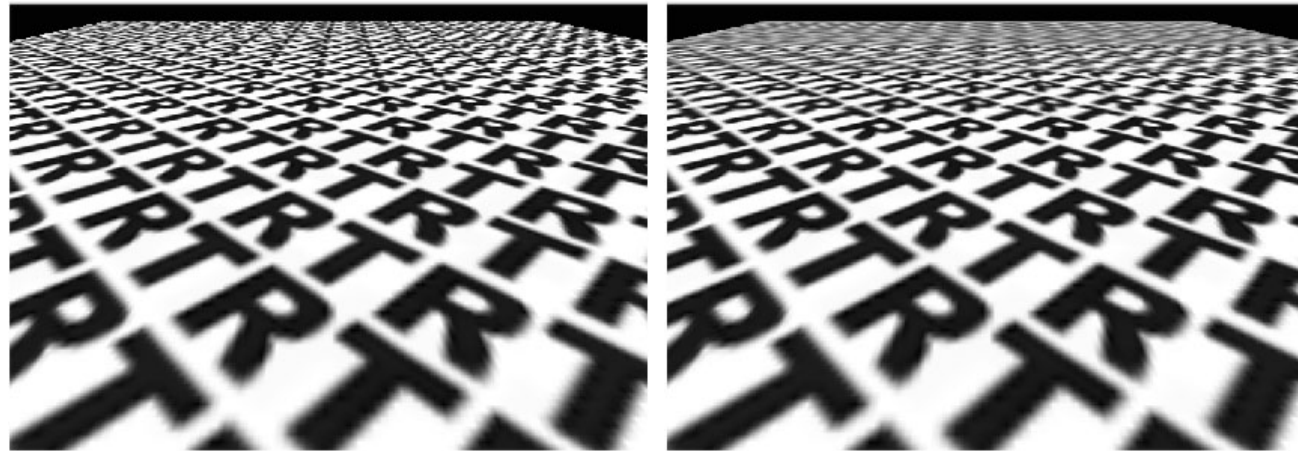
- OpenGL: `glEnable(GL_LINE_SMOOTH)`  
alternatively: `GL_POLYGON_SMOOTH`

# Aliasing

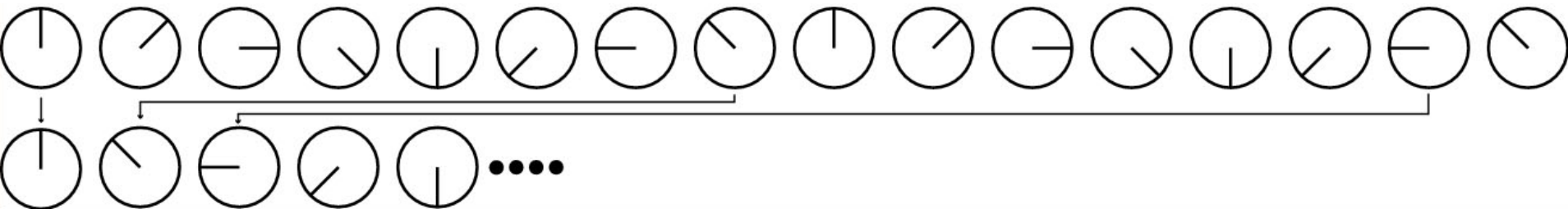
- Pixels



- Texture

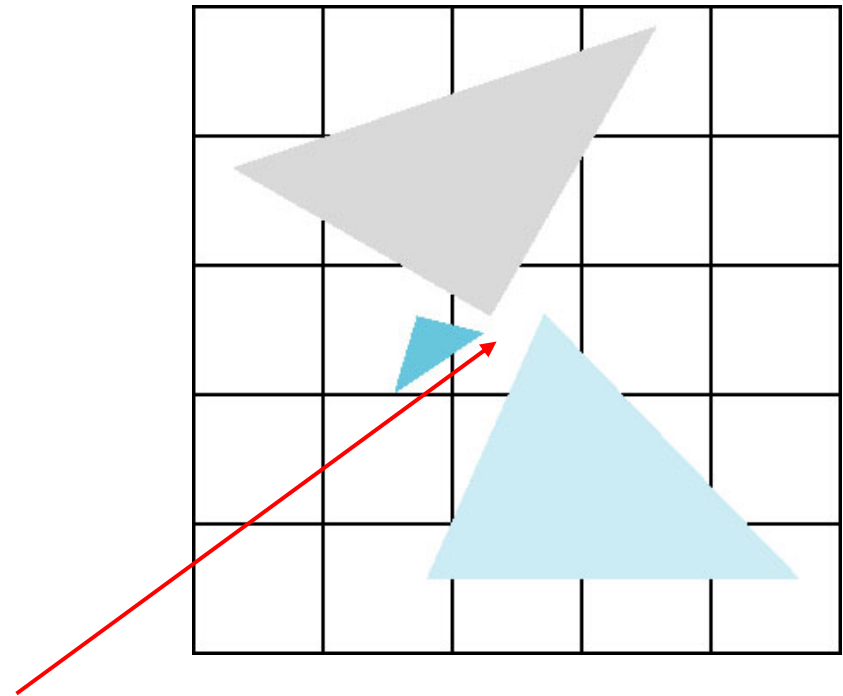


- Time



# Polygon Aliasing

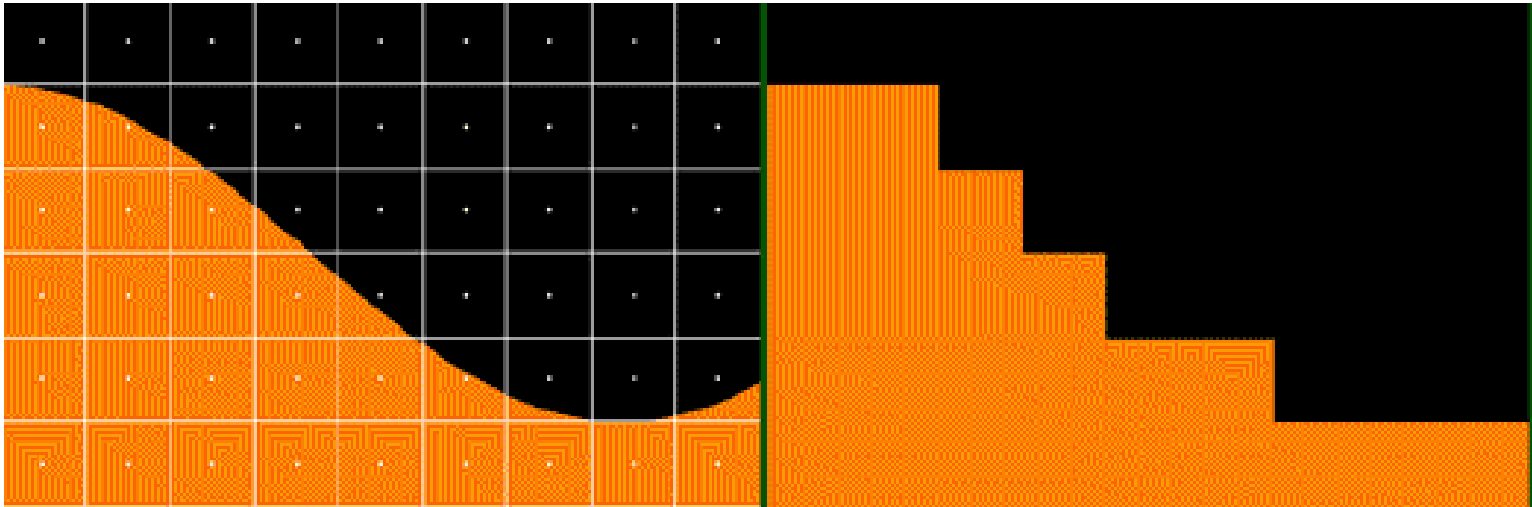
- Aliasing problems can be serious for polygons
  - Jaggedness of edges
  - Small polygons neglected
  - Need compositing so color of one polygon does not totally determine color of pixel



All three polygons should contribute to color

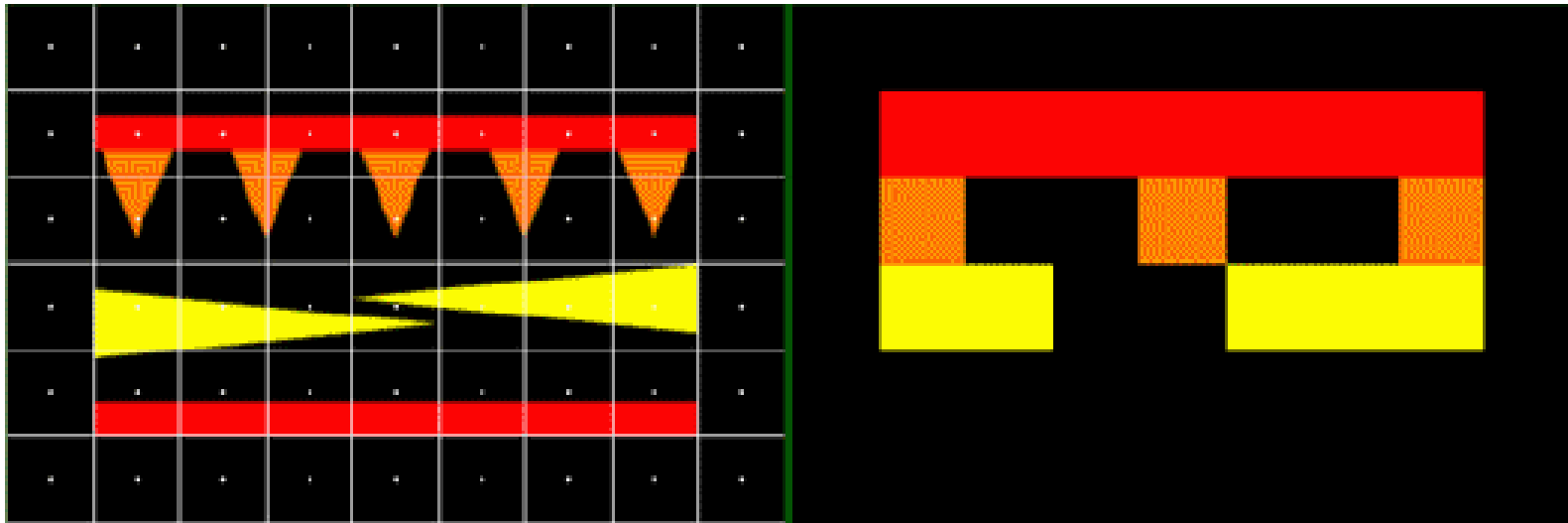
# Examples of Aliasing

Jagged boundaries



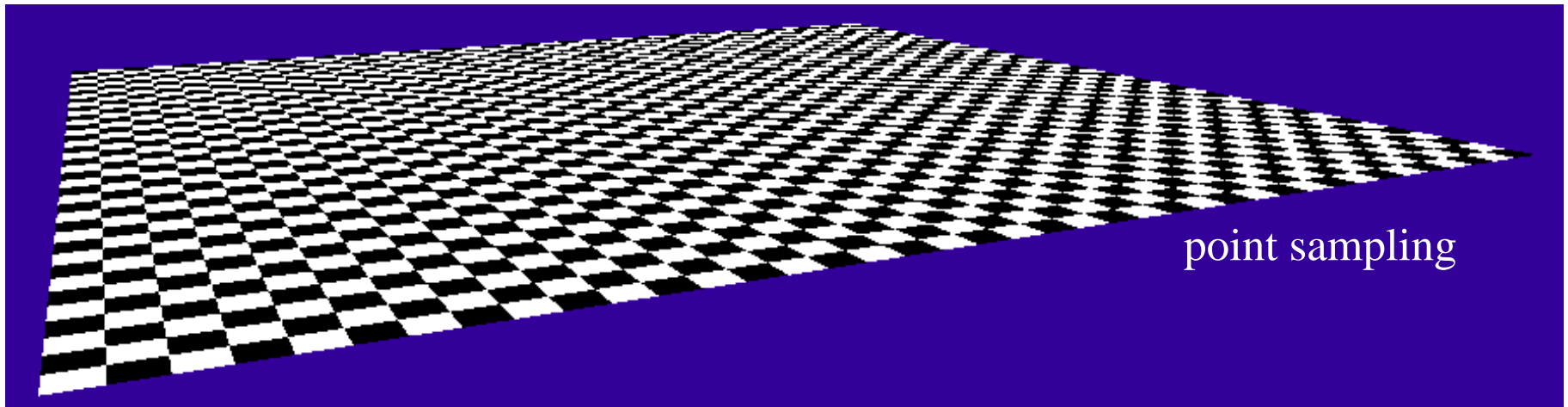
# Examples of Aliasing

Improperly rendered detail



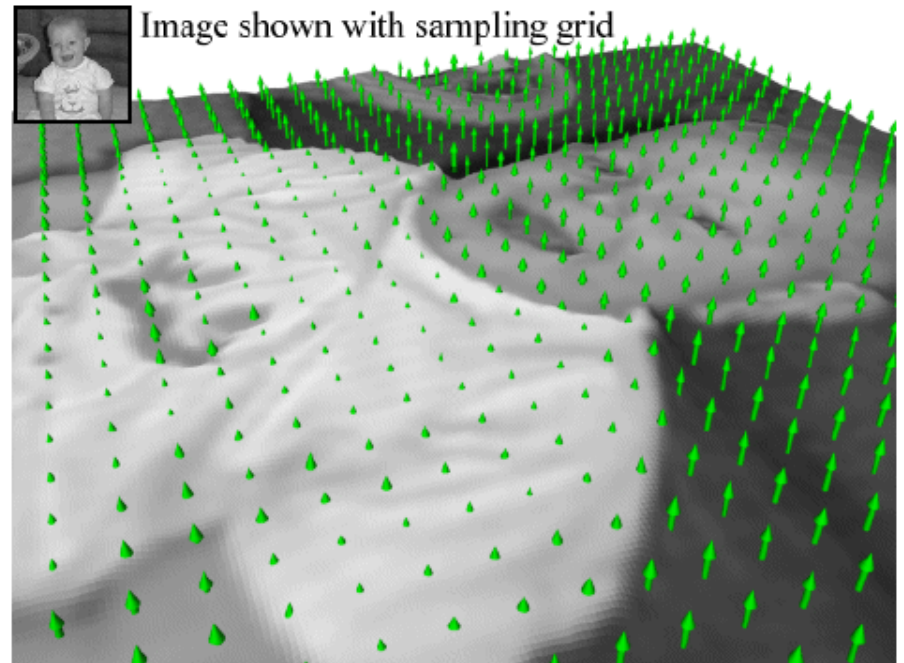
# Examples of Aliasing

## Texture Errors



# Sampling Density

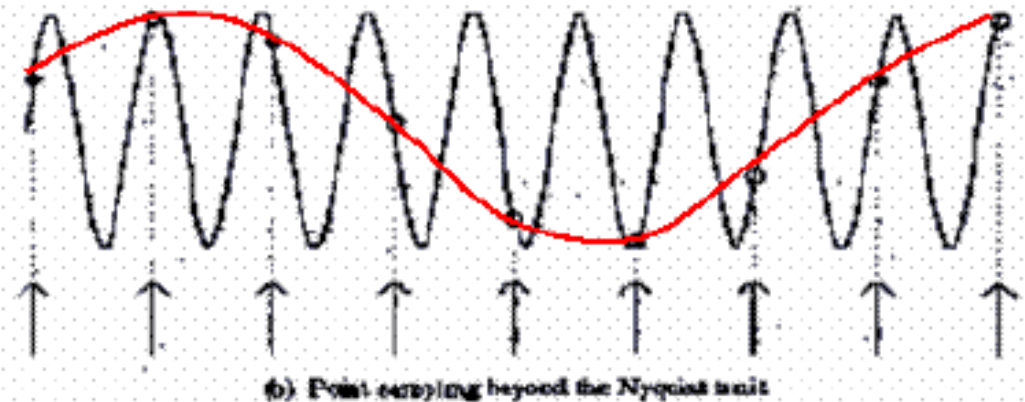
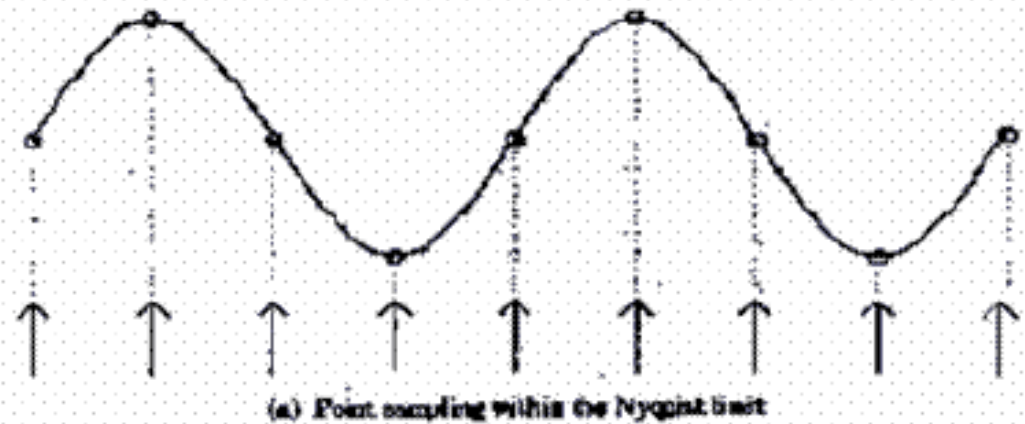
- How densely must we sample an image in order to capture its essence?
- If we under-sample the signal, we won't be able to accurately reconstruct it...



# Sampling Density

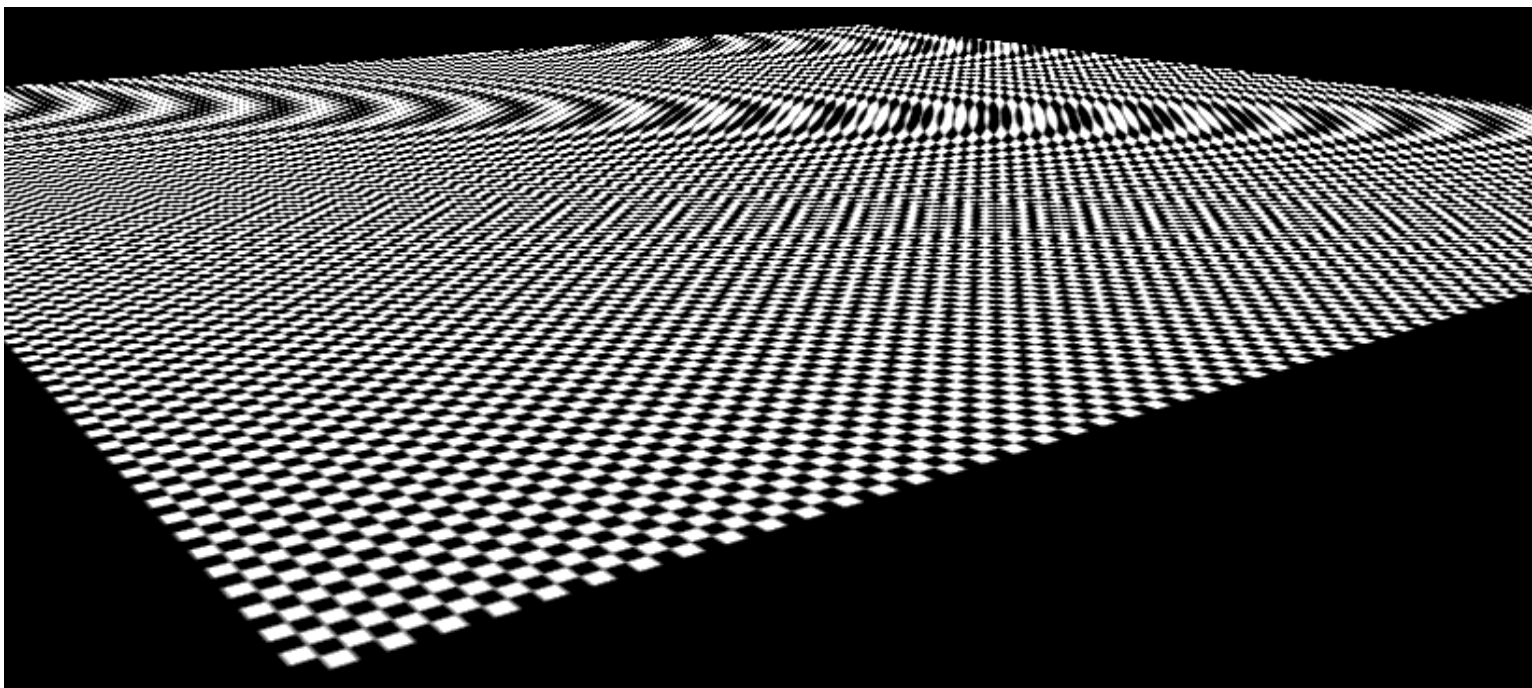
- If we insufficiently sample the signal, it may be mistaken for something simpler during reconstruction (that's aliasing!)

Image from Robert L. Cook,  
 "Stochastic Sampling and  
 Distributed Ray Tracing",  
 An Introduction to Ray Tracing,  
 Andrew Glassner, ed.,  
 Academic Press Limited, 1989.



# Sampling Density

- Aliasing in 2D because of insufficient sampling density

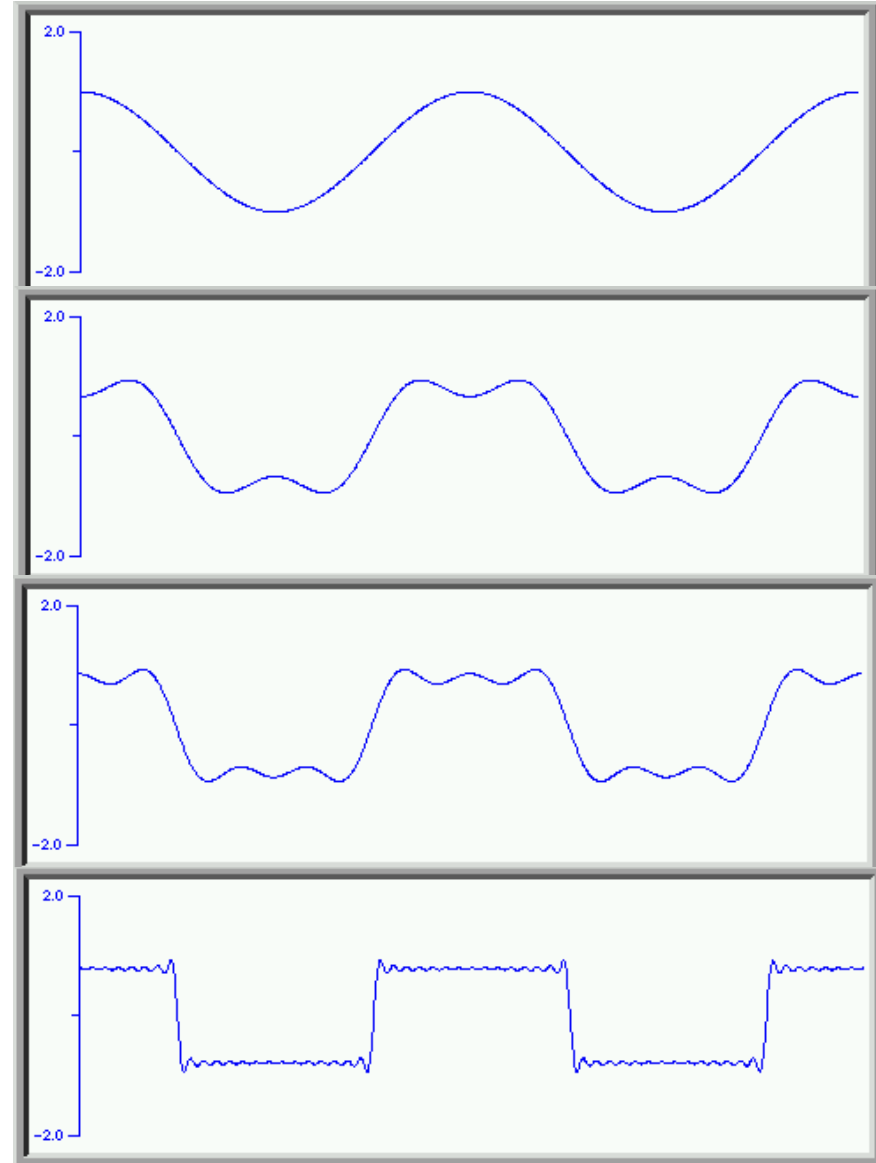


# Remember Fourier Analysis?

- All periodic signals can be represented as a summation of sinusoidal waves.

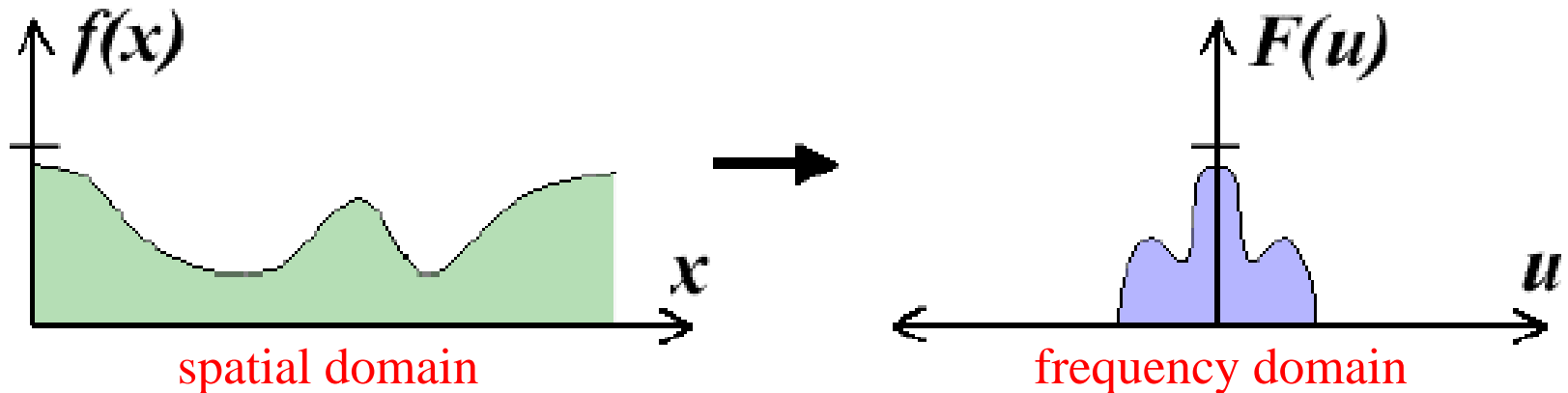
Images from

<http://axion.physics.ubc.ca/341-02/fourier/fourier.html>



# Remember Fourier Analysis?

- Every periodic signal in the *spatial domain* has a dual in the *frequency domain*.

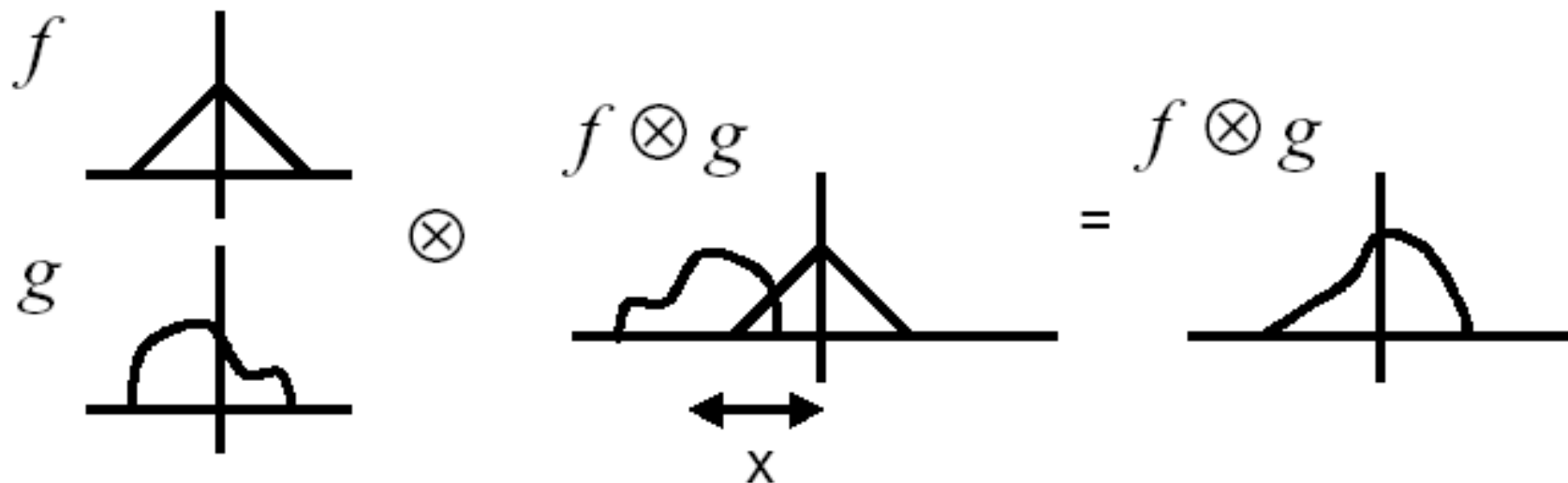


- This particular signal is *band-limited*, meaning it has no frequencies above some threshold

# Remember Convolution?

Convolution describes how a system with impulse response,  $h(x)$ , reacts to a signal,  $f(x)$ .

$$f(x) * h(x) = \int_{-\infty}^{\infty} f(\lambda)h(x - \lambda)d\lambda$$



CS174 Fall 99 Lecture 7

Copyright © Mark Meyer

Images from Mark Meyer  
<http://www.gg.caltech.edu/~cs174ta/>

# Remember Convolution?

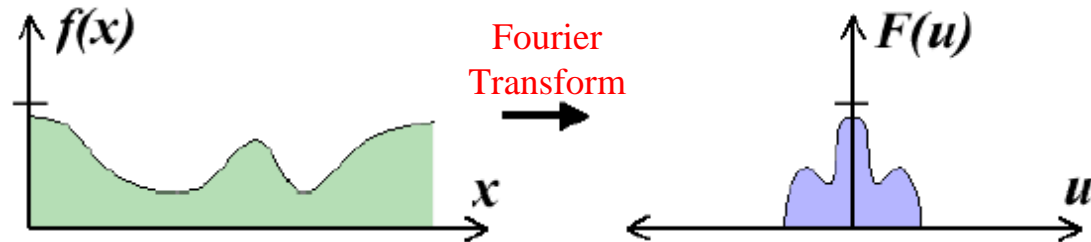
- Some operations that are difficult to compute in the spatial domain can be simplified by transforming to its dual representation in the frequency domain.
- For example, convolution in the spatial domain is the same as multiplication in the frequency domain.
- And, convolution in the frequency domain is the same as multiplication in the spatial domain

$$f(x) * h(x) \rightarrow F(u)H(u)$$

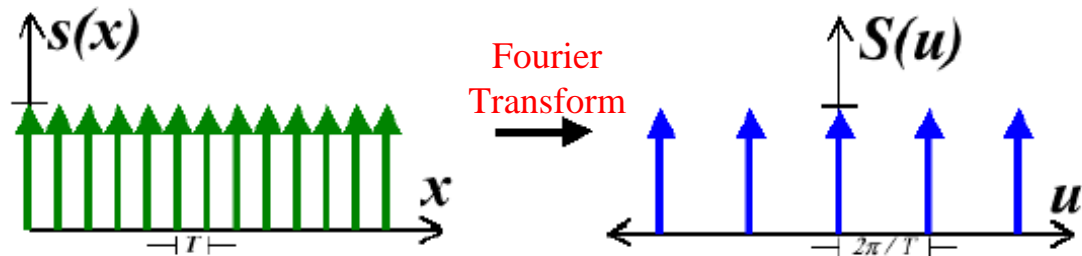
$$F(u) * H(u) \rightarrow f(x)h(x)$$

# Sampling in the Frequency Domain

original  
signal



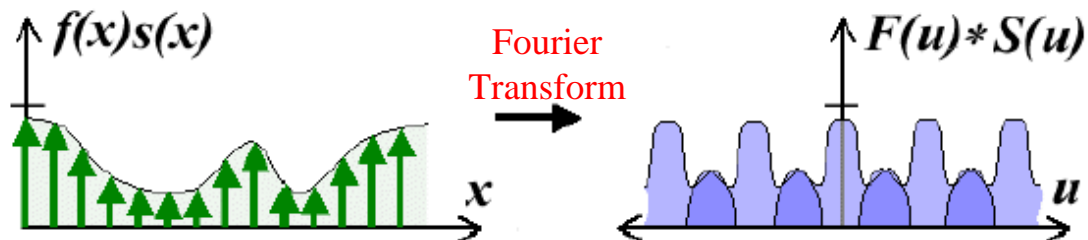
sampling  
grid



(multiplication)

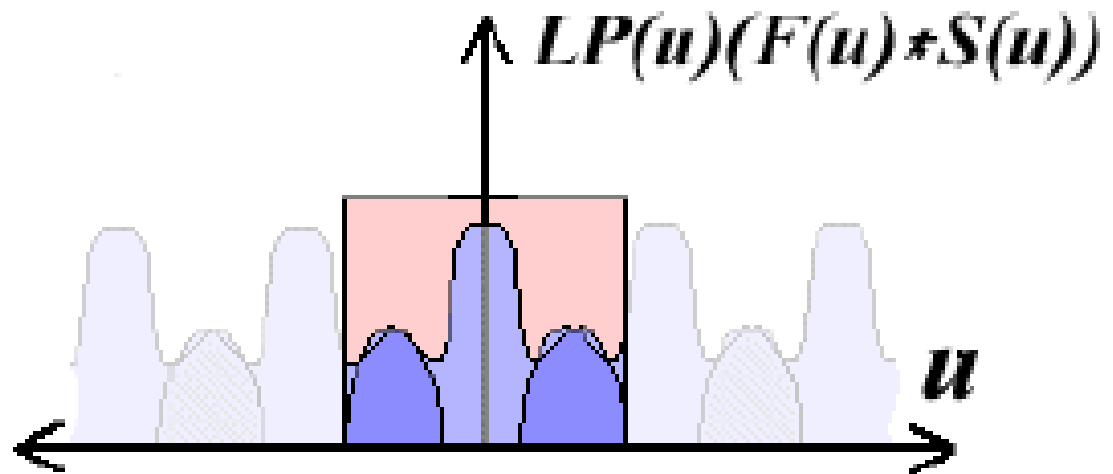
(convolution)

sampled  
signal



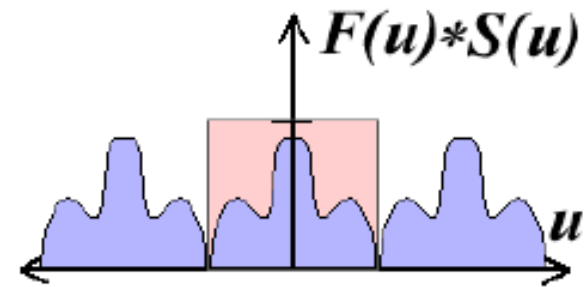
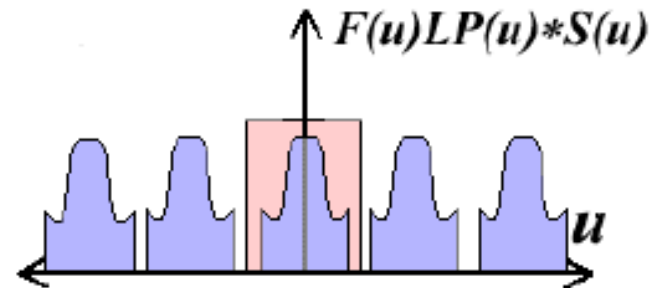
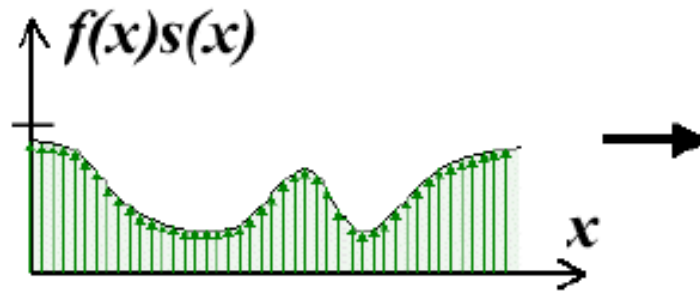
# Reconstruction

- If we can extract a copy of the original signal from the frequency domain of the sampled signal, we can reconstruct the original signal!
- But there may be overlap between the copies.



# Guaranteeing Proper Reconstruction

- Separate by removing high frequencies from the original signal (low pass pre-filtering)
- Separate by increasing the sampling density



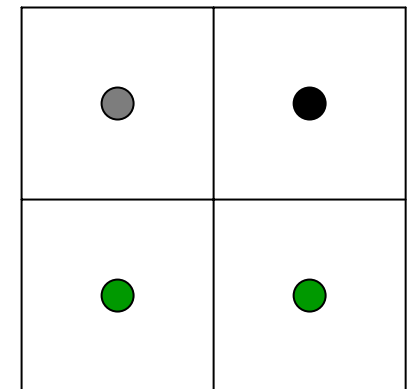
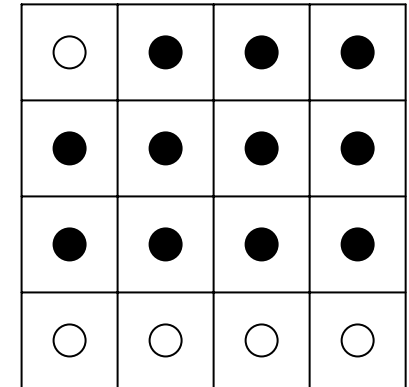
- If we can't separate the copies, we will have overlapping frequency spectrum during reconstruction  $\rightarrow$  *aliasing*.

# Sampling Theorem

- When sampling a signal at discrete intervals, the sampling frequency must be *greater than twice* the highest frequency of the input signal in order to be able to reconstruct the original perfectly from the sampled version (Shannon, Nyquist)

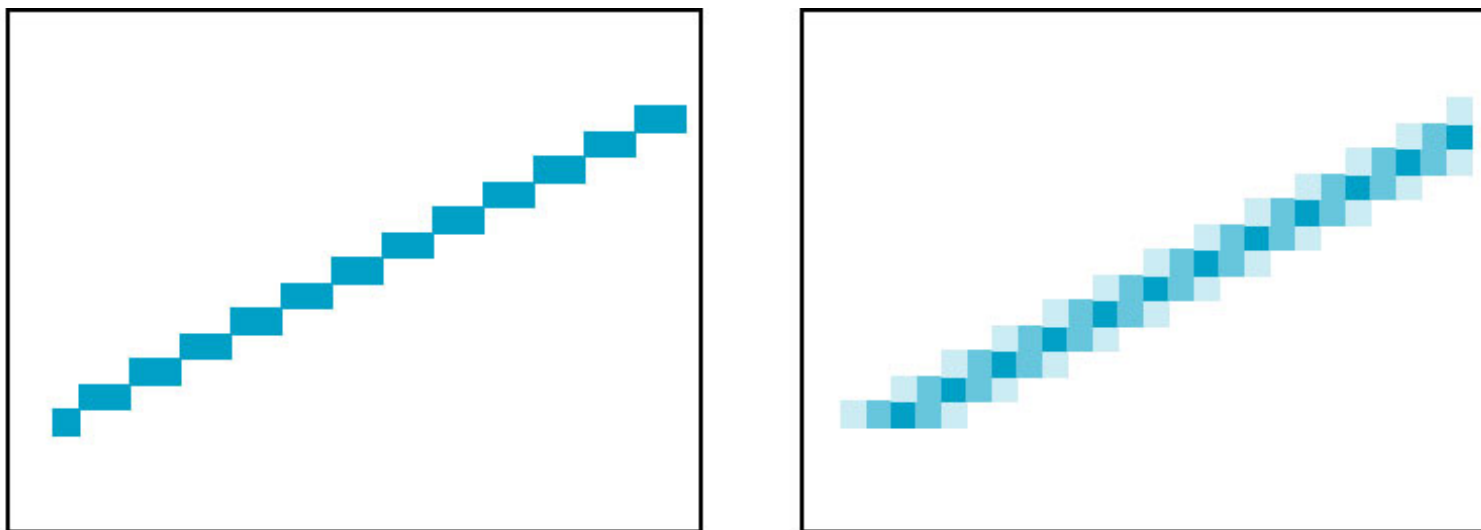
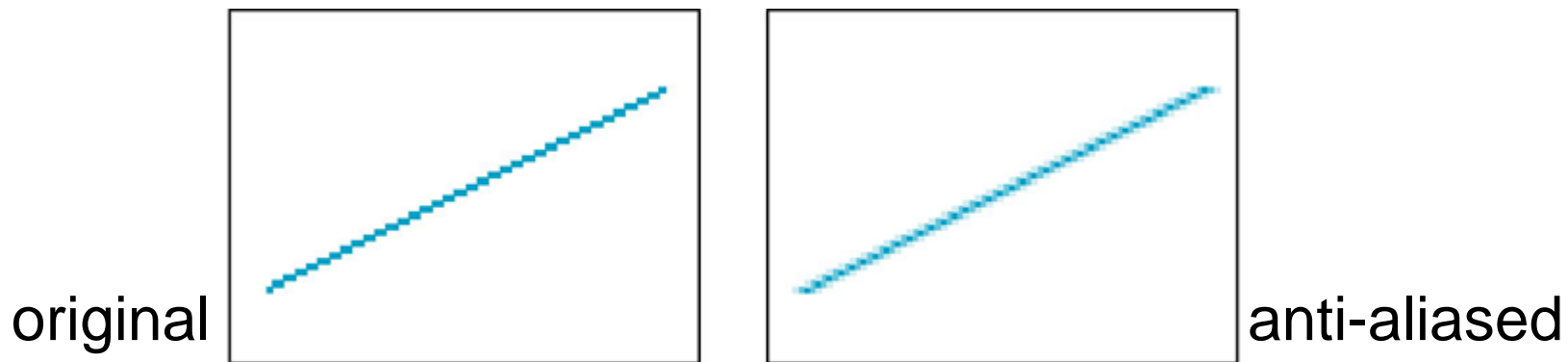
# Antialiasing

- Application of techniques to reduce/eliminate aliasing artifacts
- Some of the methods are
  - increasing sampling rate by increasing the resolution. Display memory requirements increases four times if the resolution is doubled
  - averaging methods (post processing). Intensity of a pixel is set as the weighted average of its own intensity and the intensity of the surrounding pixels
  - Area sampling, more popular



2x2 sampling per pixel,  
with simple averaging  
to get final intensity

# Anti-aliasing by Area Averaging



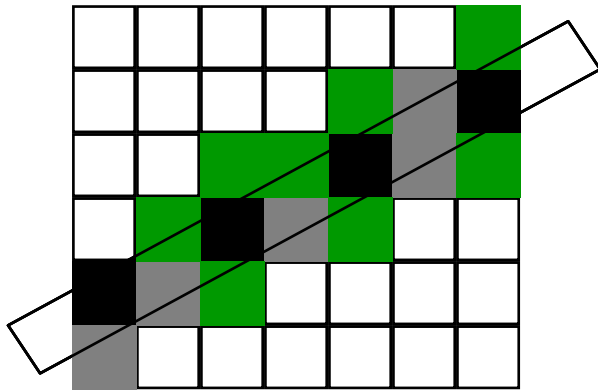
magnified

# Antialiasing

- Application of techniques to reduce/eliminate aliasing artifacts
- Some of the methods are:
  - increasing sampling rate by increasing the resolution. Display memory requirements increase four times if the resolution is doubled
  - averaging methods (post processing). Intensity of a pixel is set as the weighted average of its own intensity and the intensity of the surrounding pixels
  - area sampling, more popular

# Area Sampling

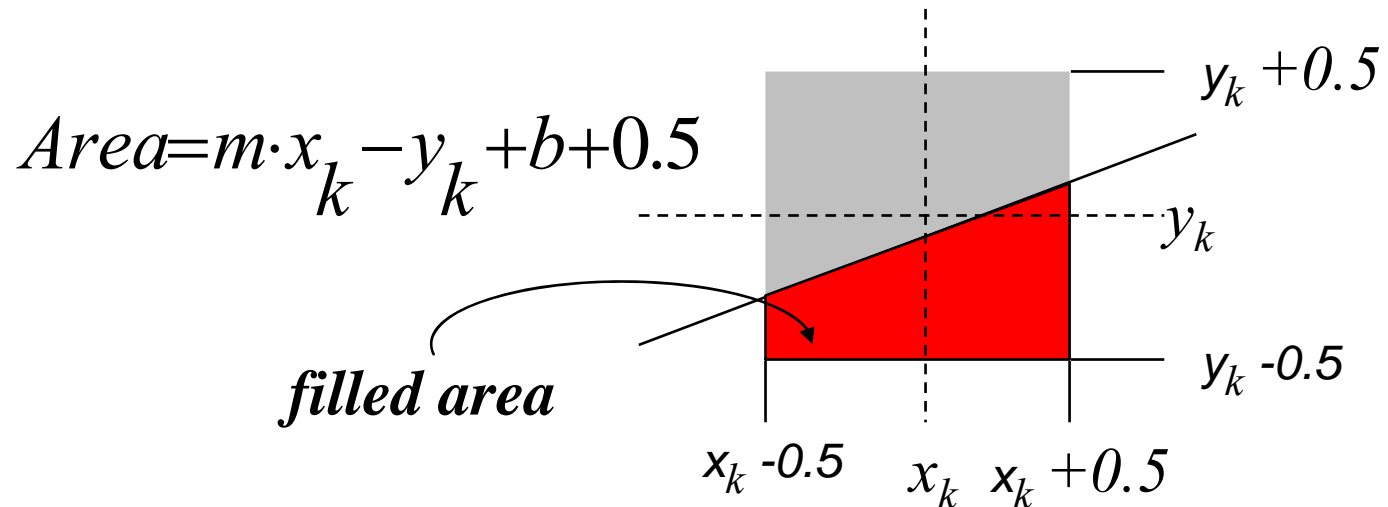
- A scan converted primitive occupies finite area on the screen
- Intensity of the boundary pixels is adjusted depending on the percent of the pixel area covered by the primitive. This is called *weighted area sampling*
- Intensities can be weighted depending on the distance of the area from the center of the pixel



0	0	0	1/8	0
0	0	1/4	.914	1/8
0	1/4	.914	1/4	0
1/8	.914	1/4	0	0
0	1/8	0	0	0

# Area Sampling

- Methods to estimate percent of pixel covered by the primitive
  - subdivide pixel into sub-pixels and determine how many sub-pixels are inside the boundary
  - incremental line algorithm can be extended, with area calculated as



- efficient only when one edge of the area passes through the pixel

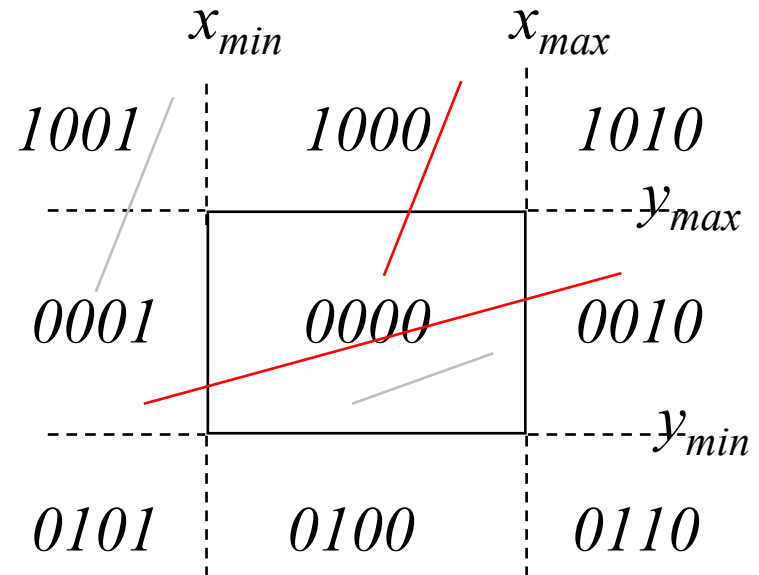
# Clipping

- Clipping of primitives is done usually before scan converting the primitives
- Reasons being
  - scan conversion needs to deal only with the clipped version of the primitive, which might be much smaller than its unclipped version
  - primitives are usually defined in the real world, and their mapping from the real to the integer domain of the display might result in the overflowing of the integer values resulting in unnecessary artifacts



# Cohen-Sutherland Line-Clipping Algorithm

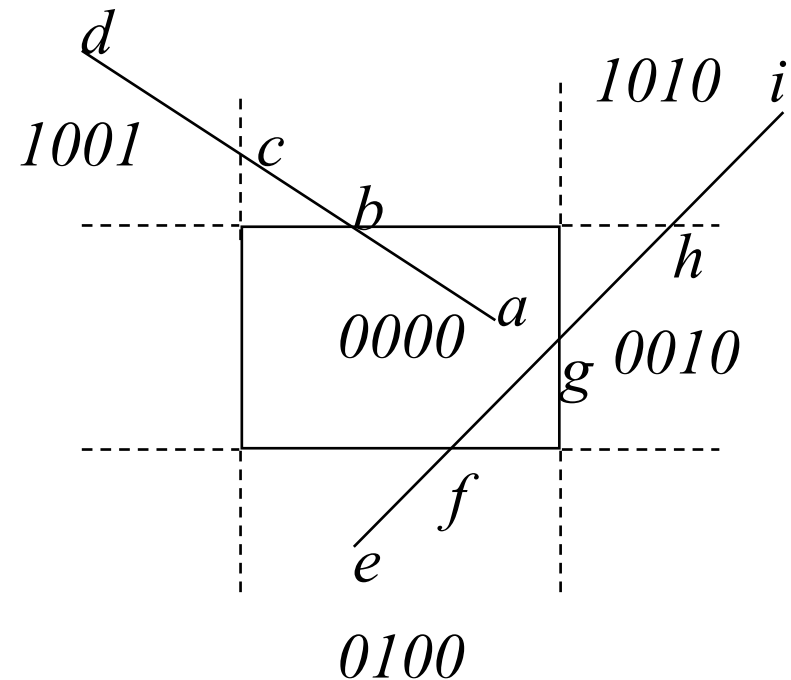
- Extend the edges of the clip rectangle to divide its plane into nine regions
- Each region is assigned a 4-bit code, determined by where the region lies w.r.t. the outside half-planes of the clip-rectangle edges
- Trivial rejection –  
If the logical *and* of the 4-bit codes of the end points is non-zero, then reject.



$bit-1 \quad y > y_{max}$ ,  $bit-2 \quad y < y_{min}$   
 $bit-3 \quad x > x_{max}$ ,  $bit-4 \quad x < x_{min}$

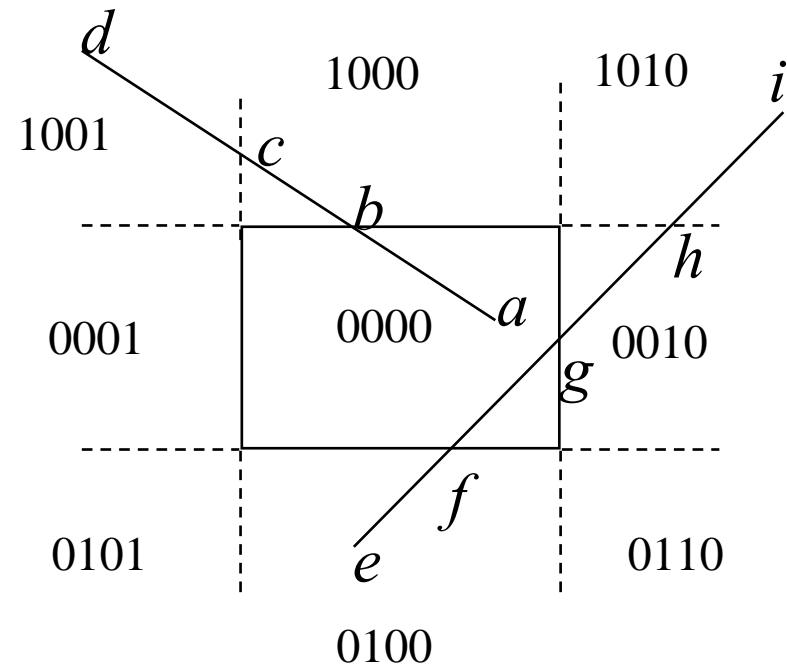
# Cohen-Sutherland Line-Clipping Algorithm

- To do the clipping, find the end point that lies outside
- Test the outcode to find the edge that is crossed and determine the corresponding intersection point. Look for rightmost 1.
- Replace the outside end-point by intersection-point and update its outcode.
- If the logical *and* of the outcodes of the new and old end points is non-zero, then reject. If the outcodes of the new and old end points are both 0000, then accept .
- Repeat the above steps for the new line segment.



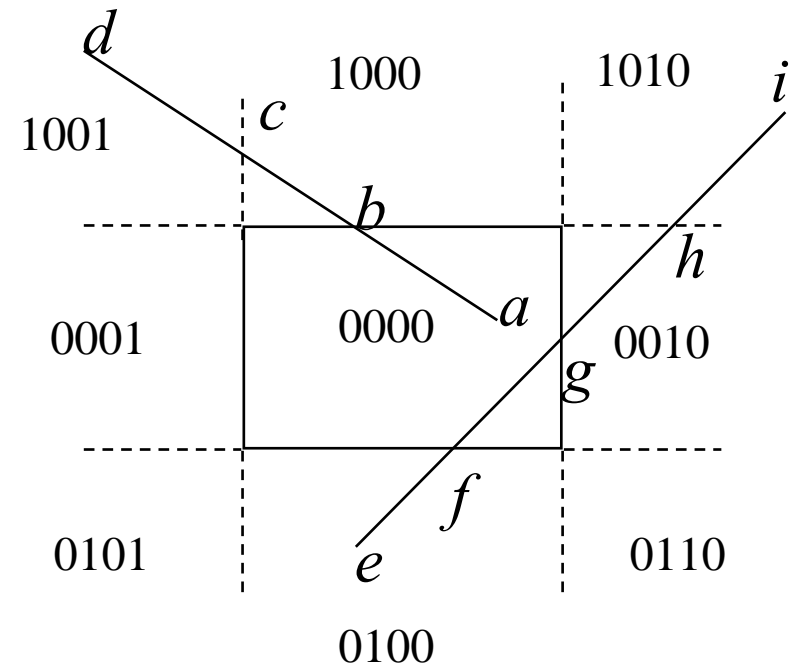
# Cohen-Sutherland Line-Clipping Algorithm-Example

- Consider line segment  $ei$ . The outcodes of endpoints are  $c(e)=0100$ ,  $c(i)=1010$ . The logical AND of the outcodes is zero and the outcodes are not both 0000, so continue.
- Pick one of the endpoints, e.g.,  $e$ . Test the outcode of  $e$  to find the rightmost 1. It is the 3<sup>rd</sup> bit which represents the bottom clip region so we clip against the bottom clip edge yielding the new endpoint  $f$  with code 0000. The logical AND of the outcodes is zero and the outcodes are not both 0000, so continue. We are done with the left end.
- Pick  $i$ . Test the outcode of  $i$  to find the rightmost 1. It is the 2<sup>nd</sup> bit which represents the right clip region so we clip against right clip edge yielding the new endpoint  $g$  with code 0000. The logical AND of the outcodes is zero and the outcodes are both 0000, so stop and output the clipped line segment  $fg$ .

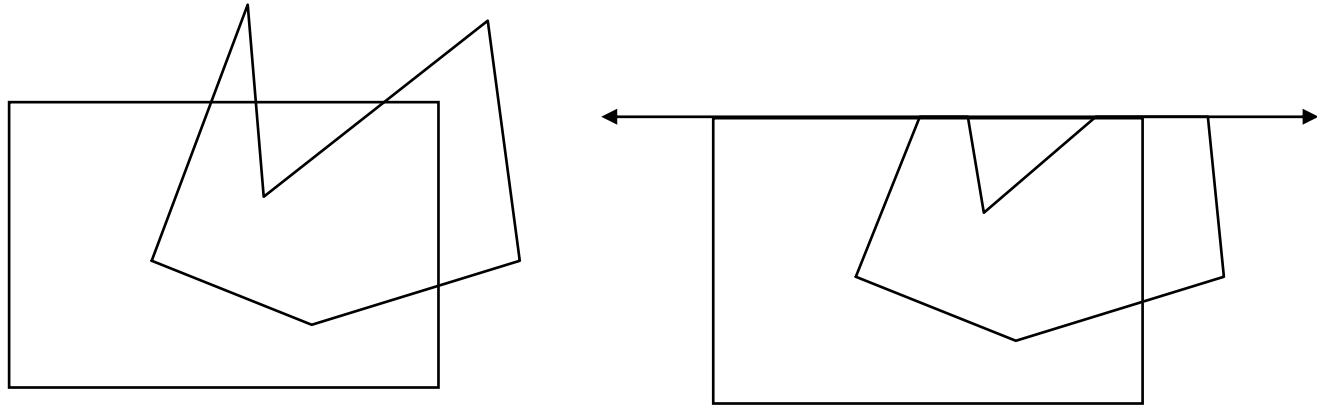


# Cohen-Sutherland Line-Clipping Algorithm-Example

- Consider line segment  $ad$ . The outcodes of endpoints are  $c(a)=0000$ ,  $c(d)=1001$ . The logical AND of the outcodes is zero and the outcodes are not both 0000, so continue.
- We pick  $d$  because outcode of  $a$  is 0000. Test the outcode of  $d$  to find the rightmost 1. It is the 1<sup>st</sup> bit which represents the left clip region so we clip against the left clip edge yielding the new endpoint  $c$  with code 1000. The logical AND of the outcodes is zero and the outcodes are not both 0000, so continue.
- Pick  $c$ . Test the outcode of  $c$  to find the rightmost 1. It is the 4<sup>th</sup> bit which represents the top clip region so we clip against the top clip edge yielding the new endpoint  $b$  with code 0000. The logical AND of the outcodes is zero and the outcodes are both 0000, so stop and output the clipped line segment  $ab$ .

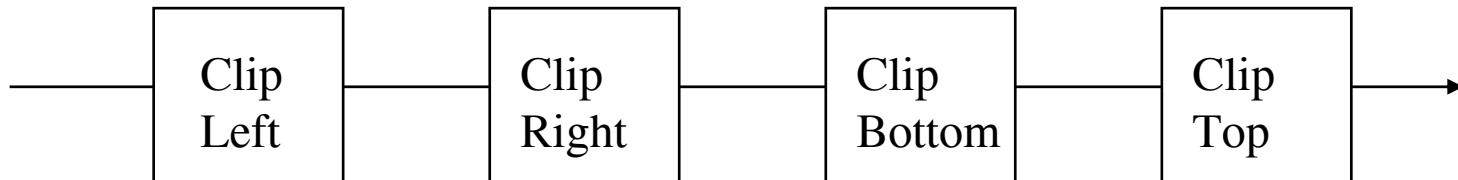


# Sutherland-Hodgeman Polygon-Clipping Algorithm



- Polygons can be clipped against each edge of the window one edge at a time. Window/edge intersections, if any, are easy to find since the X or Y coordinates are already known.
- Vertices which are kept after clipping against one window edge are saved for clipping against the remaining edges.

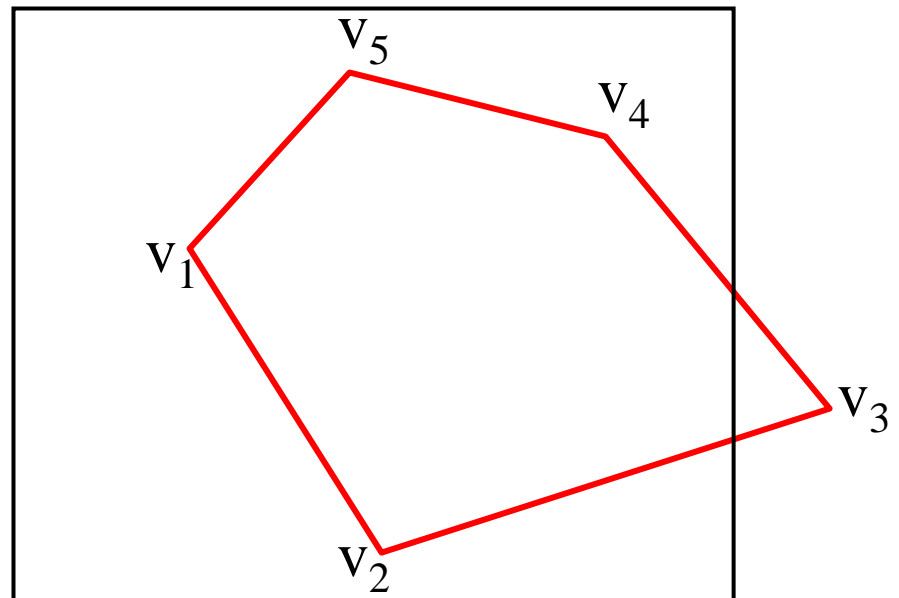
# Pipelined Polygon Clipping



- Because polygon clipping does not depend on any other polygons, it is possible to arrange the clipping stages in a **pipeline**. The input polygon is clipped against one edge and any points that are kept are passed on as input to the next *stage* of the pipeline.
- This way four polygons can be at different *stages* of the clipping process simultaneously. This is often implemented in hardware.

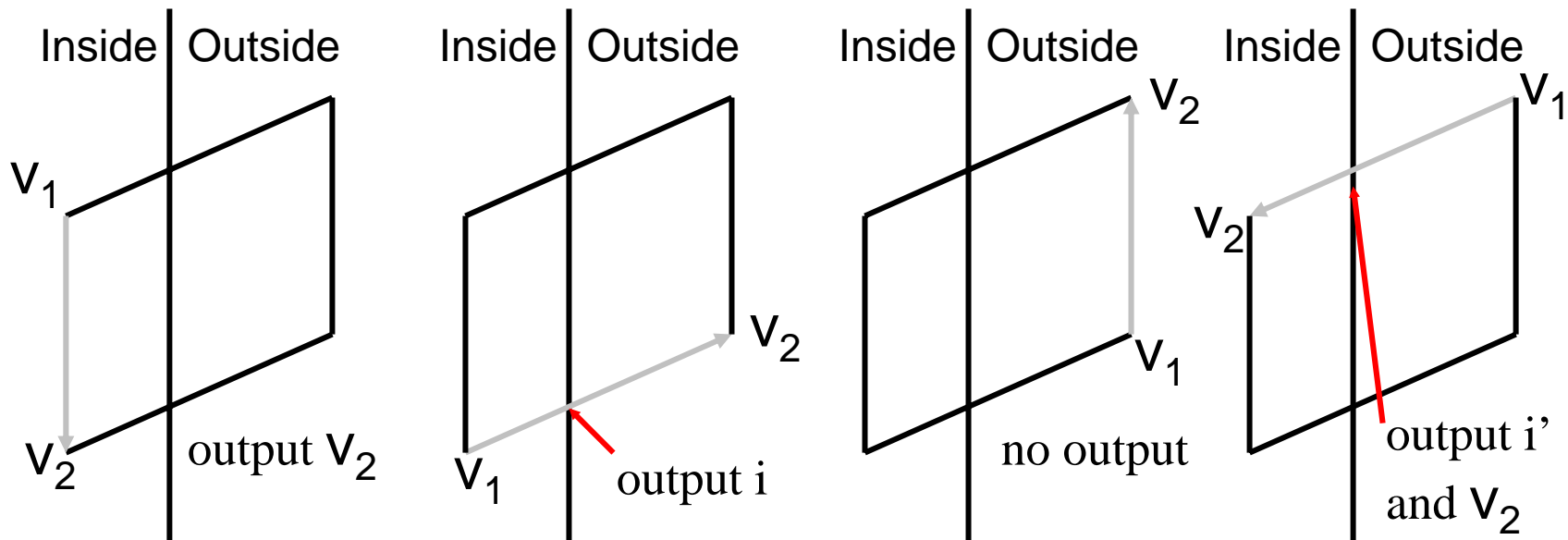
# Sutherland-Hodgeman Polygon Clipping Algorithm

- Polygon clipping is similar to line clipping except we have to keep track of inside/outside relationships
  - Consider a polygon as a list of vertices
  - Note that clipping can increase the number of vertices!
  - Typically clip one edge at a time...



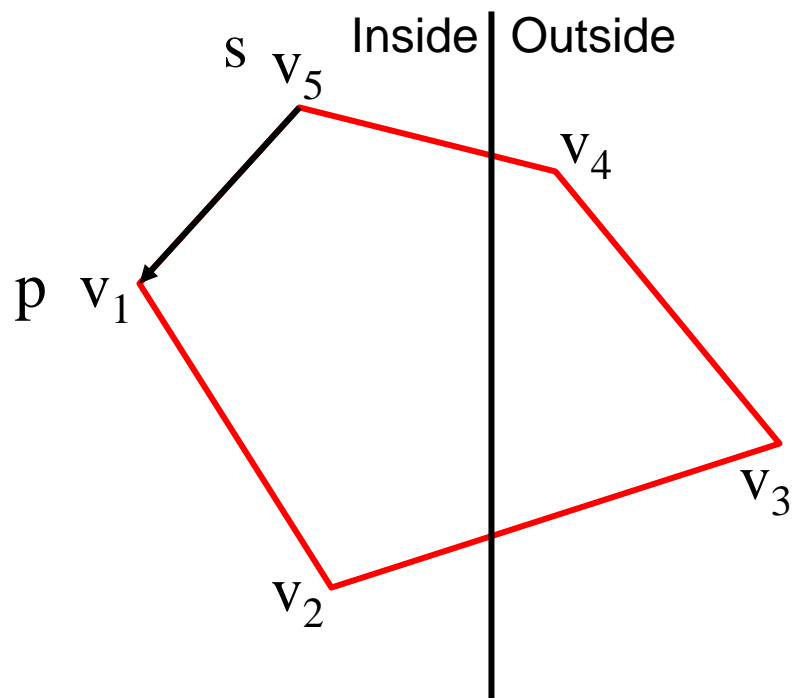
# Sutherland-Hodgeman algorithm

- Present the vertices in pairs
  - $(v_n, v_1), (v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$
  - For each pair, what are the possibilities?
  - Consider  $v_1, v_2$



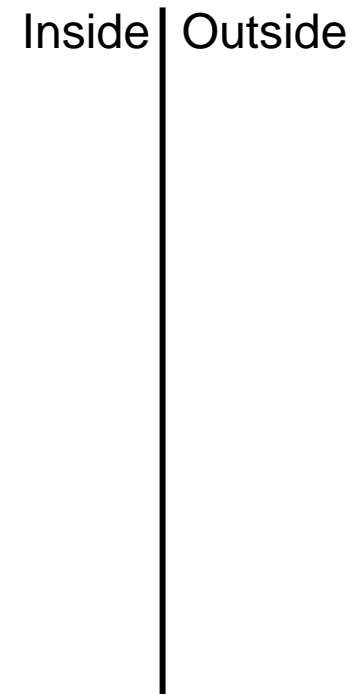
# Example

$V_5, V_1$

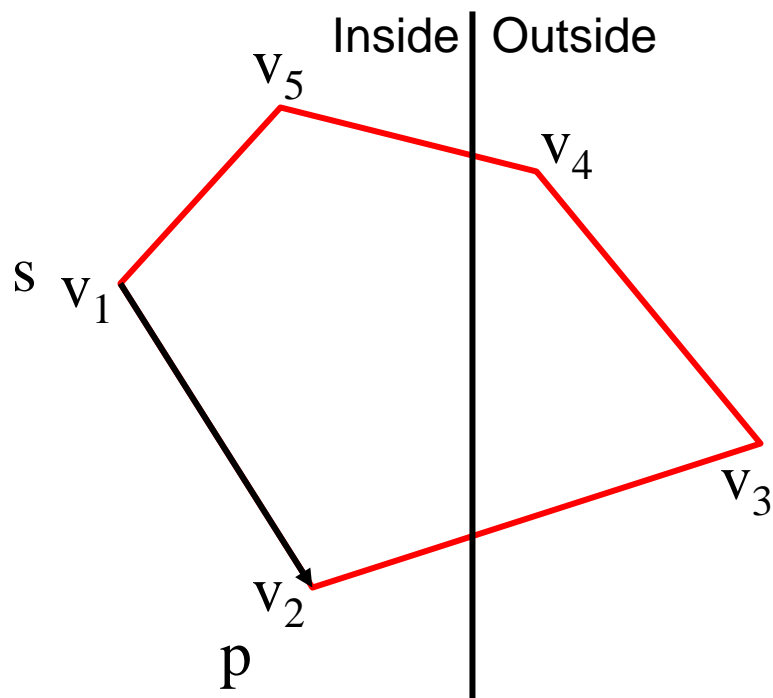


Inside, Inside  
Output  $v_1$

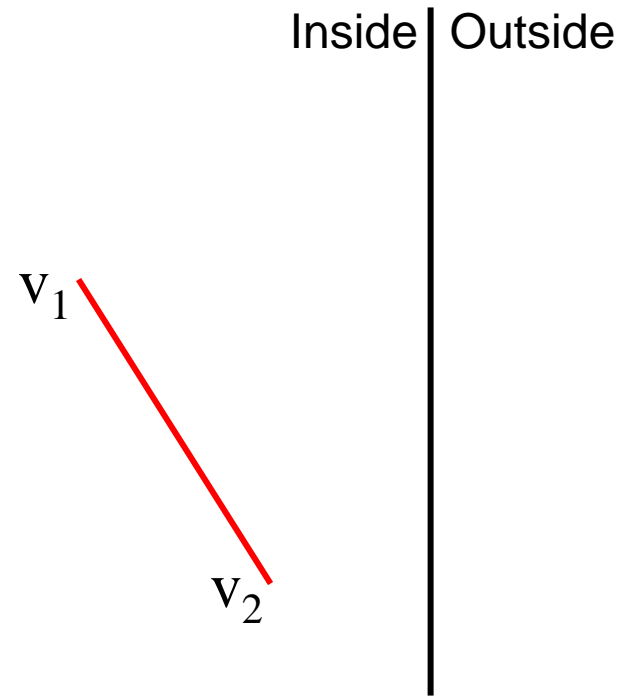
$v_1^+$



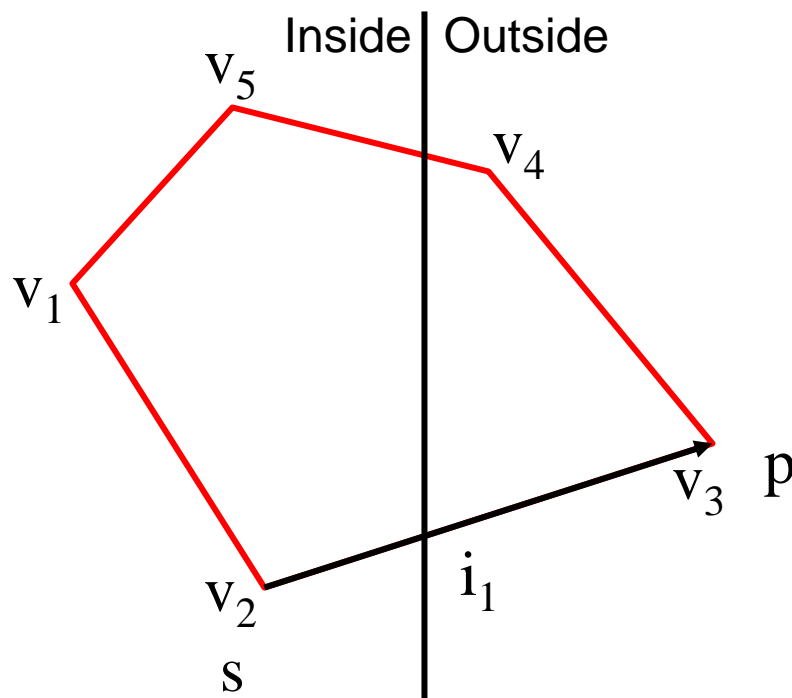
Current  
Output

$V_1, V_2$ 


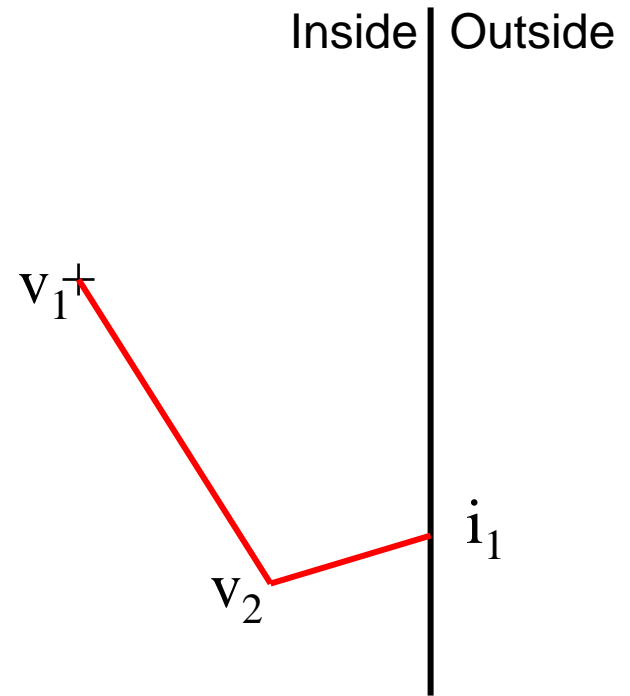
Inside, Inside  
Output  $v_2$



Current  
Output

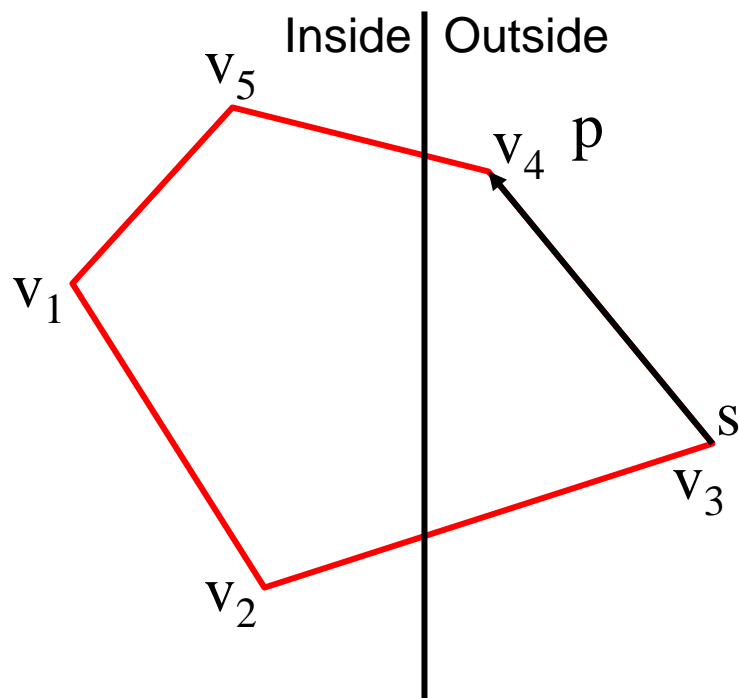
$V_2, V_3$ 


Inside, Outside  
Output  $i_1$

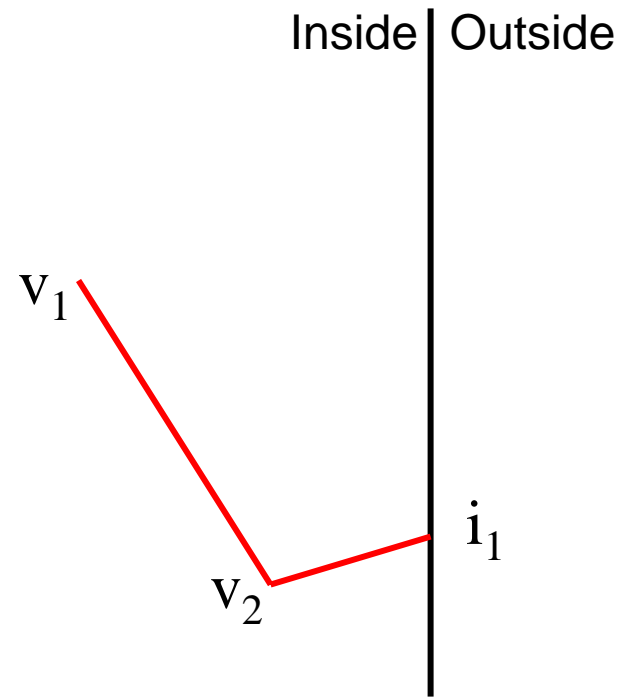


Current  
Output

$V_3, V_4$

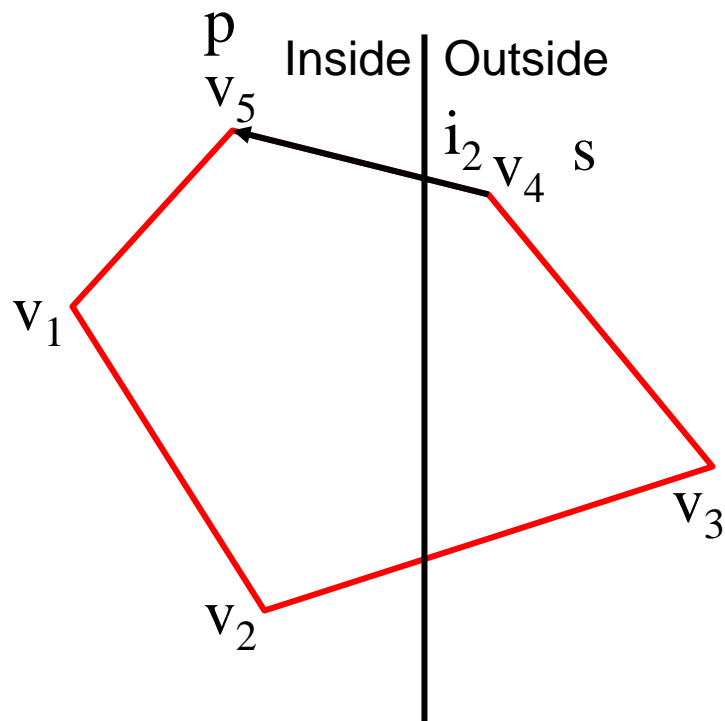


Outside, Outside  
No output

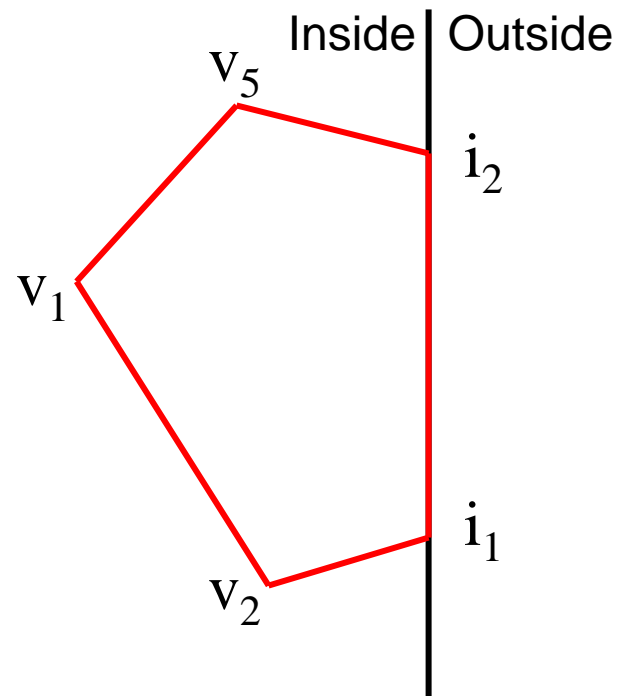


Current  
Output

$v_4, v_5$  – last edge...



Outside, Inside  
Output  $i_2, v_5$



Current  
Output