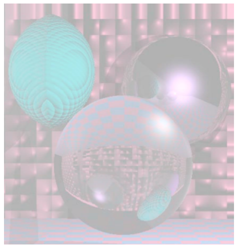


CS G140
Graduate Computer Graphics

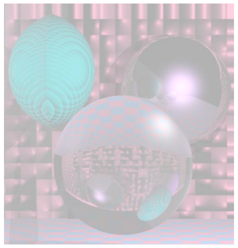
Prof. Harriet Fell

Spring 2009

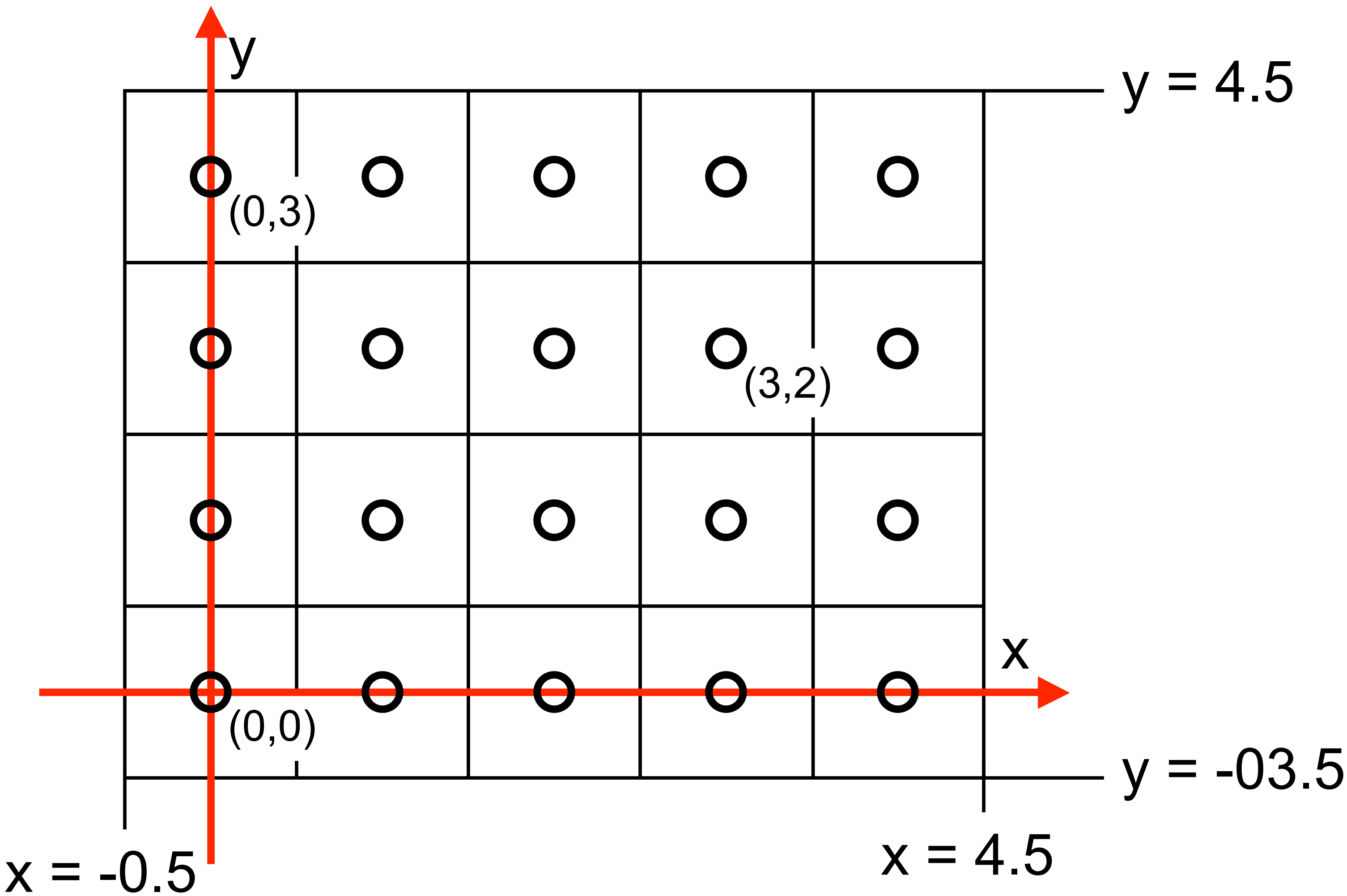
Lecture 4 – January 28, 2009

Today's Topics

- Raster Algorithms
    - Lines - Section 3.5 in Shirley *et al.*
    - Circles
    - Antialiasing
- RAY Tracing Continued
    - Ray-Plane
    - Ray-Triangle
    - Ray-Polygon



Pixel Coordinates

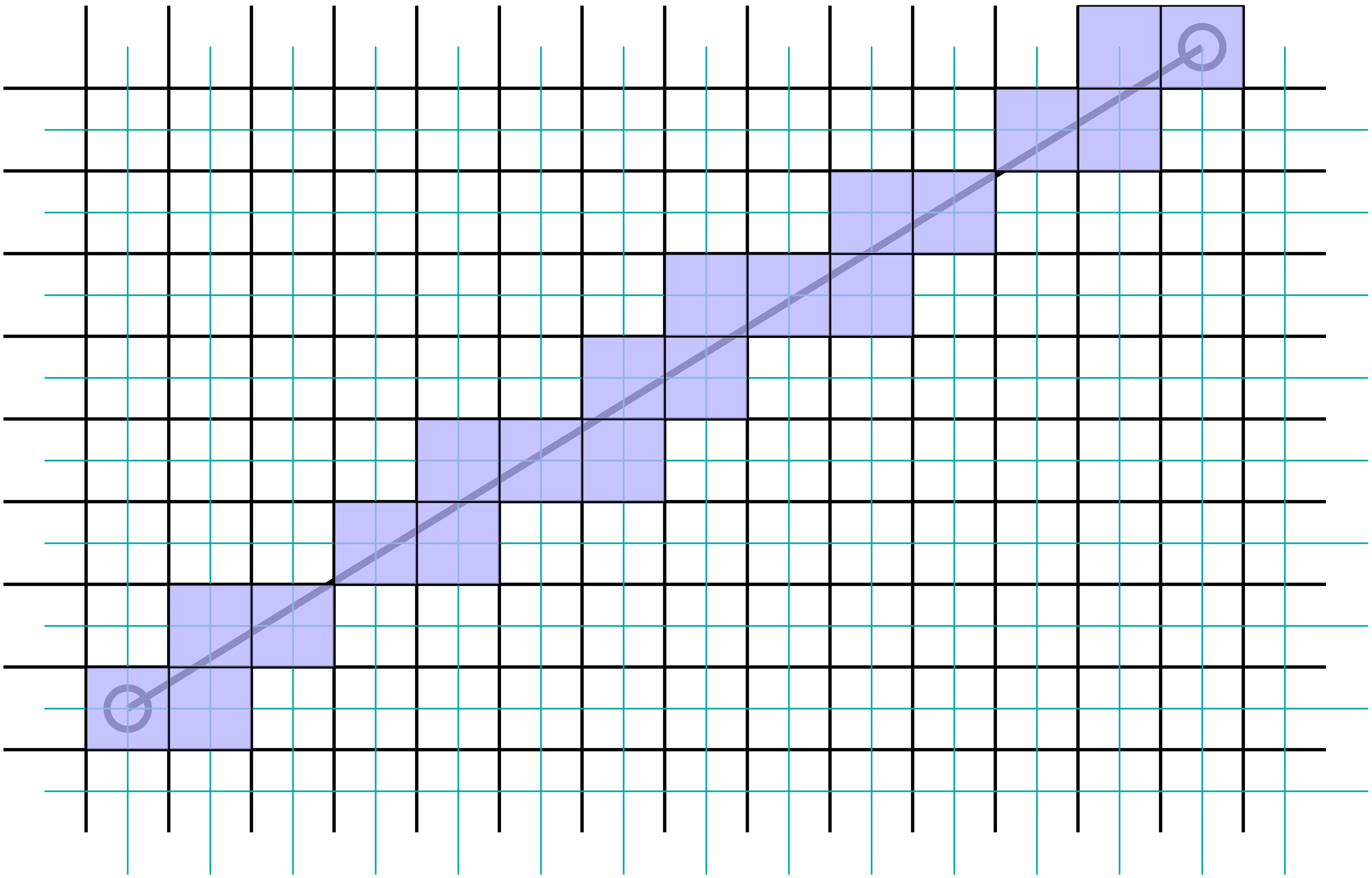


What Makes a Good Line?

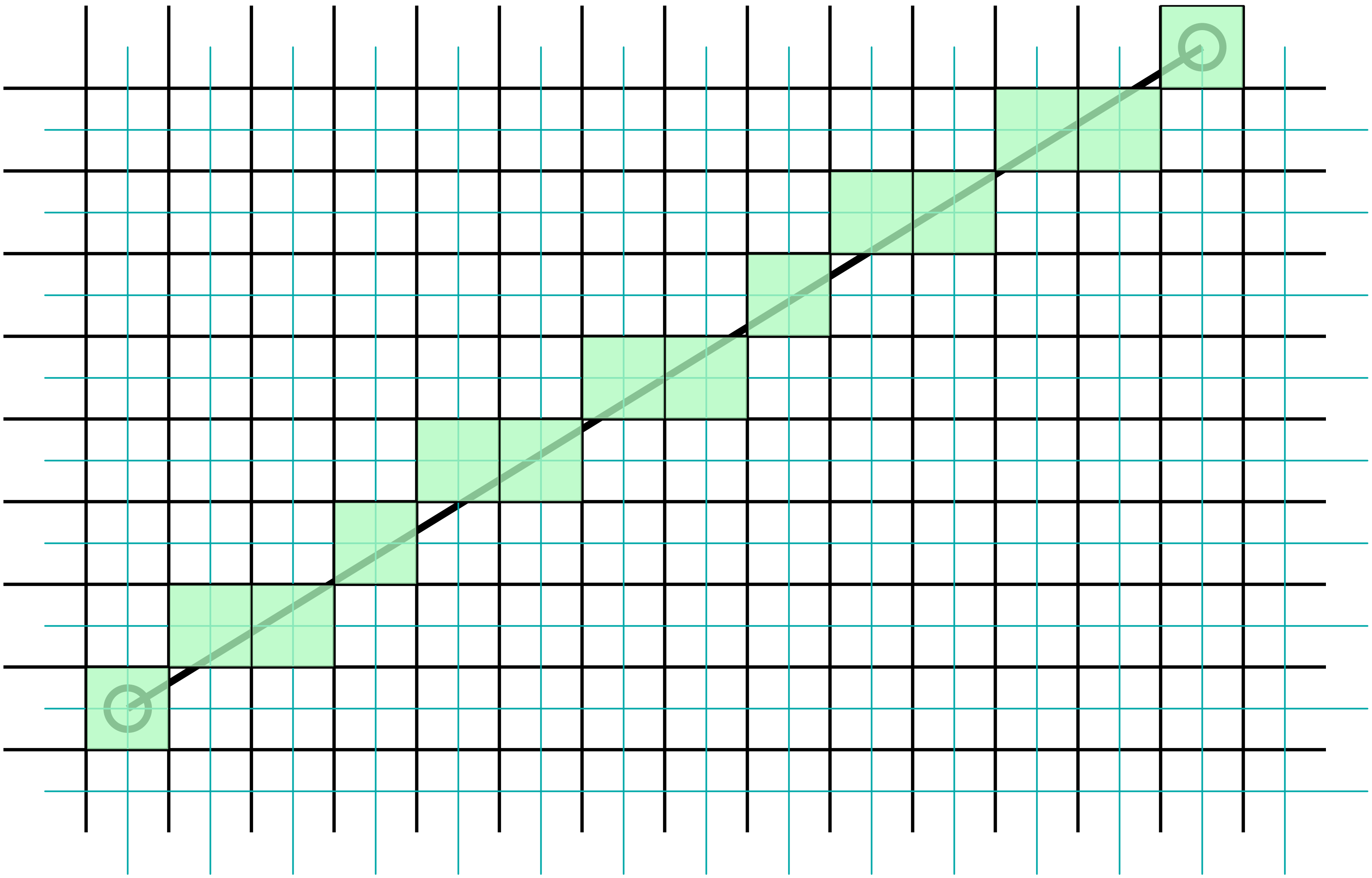
- Not too jaggy
- Uniform thickness along a line
- Uniform thickness of lines at different angles
- Symmetry, Line(P,Q) = Line(Q,P)

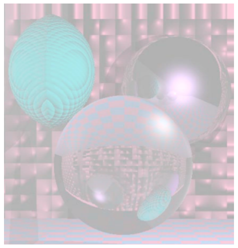
- A good line algorithm should be fast.

Line Drawing



Line Drawing





Which Pixels Should We Color?

- We could use the equation of the line:
  - $y = mx + b$
  - $m = (y_1 - y_0)/(x_1 - x_0)$
  - $b = y_1 - mx_1$
- And a loop

for $x = x_0$ to x_1

  $y = mx + b$

  draw (x, y)

This calls for real multiplication for each pixel

This only works if $x_1 \leq x_2$ and $|m| \leq 1$.

Midpoint Algorithm

- Pitteway 1967
- Van Aiken and Nowak 1985
- Draws the same pixels as the *Bresenham Algorithm* 1965.
- Uses integer arithmetic and incremental computation.
- Draws the thinnest possible line from (x_0, y_0) to (x_1, y_1) that has no gaps.
- A diagonal connection between pixels is not a gap.

Implicit Equation of a Line

$f(x,y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0$

(x_1, y_1)

$f(x,y) > 0$

$f(x,y) = 0$

$f(x,y) < 0$

(x_0, y_0)

We will assume $x_0 \leq x_1$

and that $m = (y_1 - y_0)/(x_1 - x_0)$

is in [0, 1].

Basic Form of the Algotithm

$y = y_0$

for $x = x_0$ to x_1 **do**

draw (x, y)

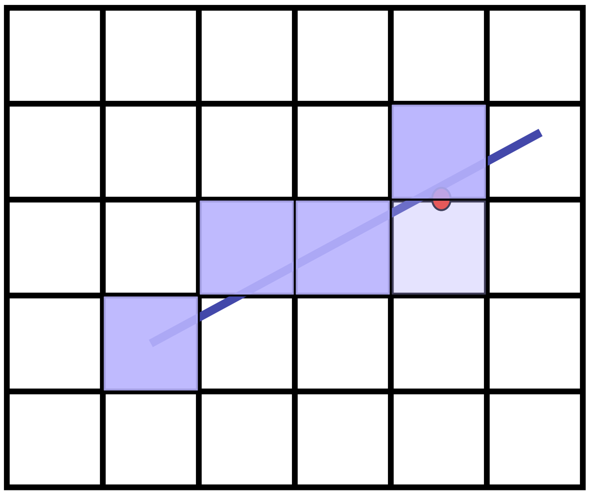
if (some condition) **then**

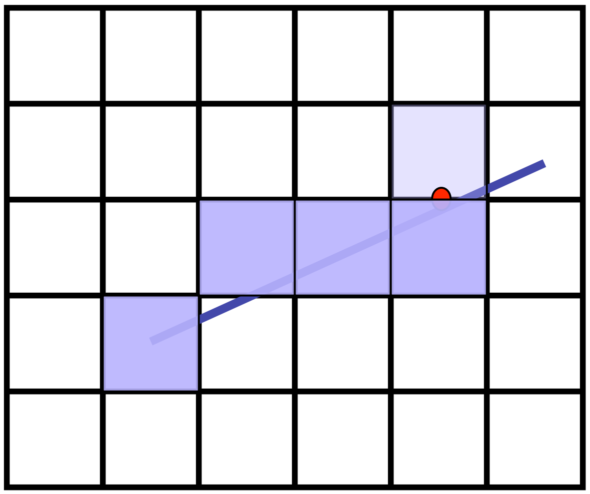
$y = y + 1$

We want to compute this condition efficiently.

Since $m \in [0, 1]$, as we move from x to x+1, the y value stays the same or goes up by 1.

Above or Below the Midpoint?





Finding the Next Pixel

Assume we just drew (x, y).

For the next pixel, we must decide between

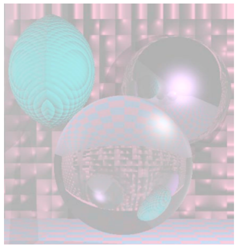
(x+1, y) and (x+1, y+1).

The midpoint between the choices is

(x+1, y+0.5).

If the line passes below (x+1, y+0.5), we draw the bottom pixel.

Otherwise, we draw the upper pixel.



The Decision Function

if $f(x+1, y+0.5) < 0$

// midpoint below line

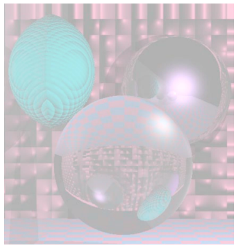
$y = y + 1$

$f(x,y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0$

How do we compute $f(x+1, y+0.5)$

incrementally?

using only integer arithmetic?



Incremental Computation

$f(x,y) = (y_0 – y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0$

$f(x + 1, y) = f(x, y) + (y_0 – y_1)$

$f(x + 1, y + 1) = f(x, y) + (y_0 – y_1) + (x_1 - x_0)$

$y = y_0$

$d = f(x_0 + 1, y + 0.5)$

for $x = x_0$ to x_1 **do**

    draw (x, y)

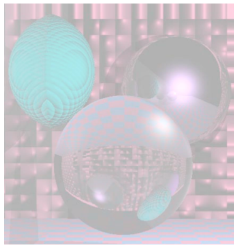
    if $d < 0$ **then**

        $y = y + 1$

        $d = d + (y_0 – y_1) + (x_1 - x_0)$

    else

        $d = d + + (y_0 – y_1)$



Integer Decision Function

$f(x,y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0$

$f(x_0 + 1, y_0 + 0.5)$

$= (y_0 - y_1)(x_0 + 1) + (x_1 - x_0)(y_0 + 0.5) + x_0 y_1 - x_1 y_0$

$2f(x_0 + 1, y_0 + 0.5)$

$= 2(y_0 - y_1)(x_0 + 1) + (x_1 - x_0)(2y_0 + 1) + 2x_0 y_1 - 2x_1 y_0$

$2f(x, y)$ = 0 if (x, y) is on the line.
< 0 if (x, y) is below the line.
> 0 if (x, y) is above the line.

Incremental Computation

$f(x,y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0$

$f(x + 1, y) = f(x, y) + (y_0 - y_1)$

$f(x + 1, y + 1) = f(x, y) + (y_0 - y_1) + (x_1 - x_0)$

$y = y_0$
$d = 2f(x_0 + 1, y + 0.5)$
for $x = x_0$ to x_1 **do**
  draw (x, y)
  if $d < 0$ **then**
    $y = y + 1$
    $d = d + 2(y_0 - y_1) + 2(x_1 - x_0)$
  else
    $d = d + + 2(y_0 - y_1)$

Midpoint Line Algorithm

$y = y_0$

$d = 2(y_0 - y_1)(x_0 + 1) + (x_1 - x_0)(2y_0 + 1) + 2x_0 y_1 - 2x_1 y_0$

for $x = x_0$ to x_1 **do**

    draw (x, y)

    if $d < 0$ **then**

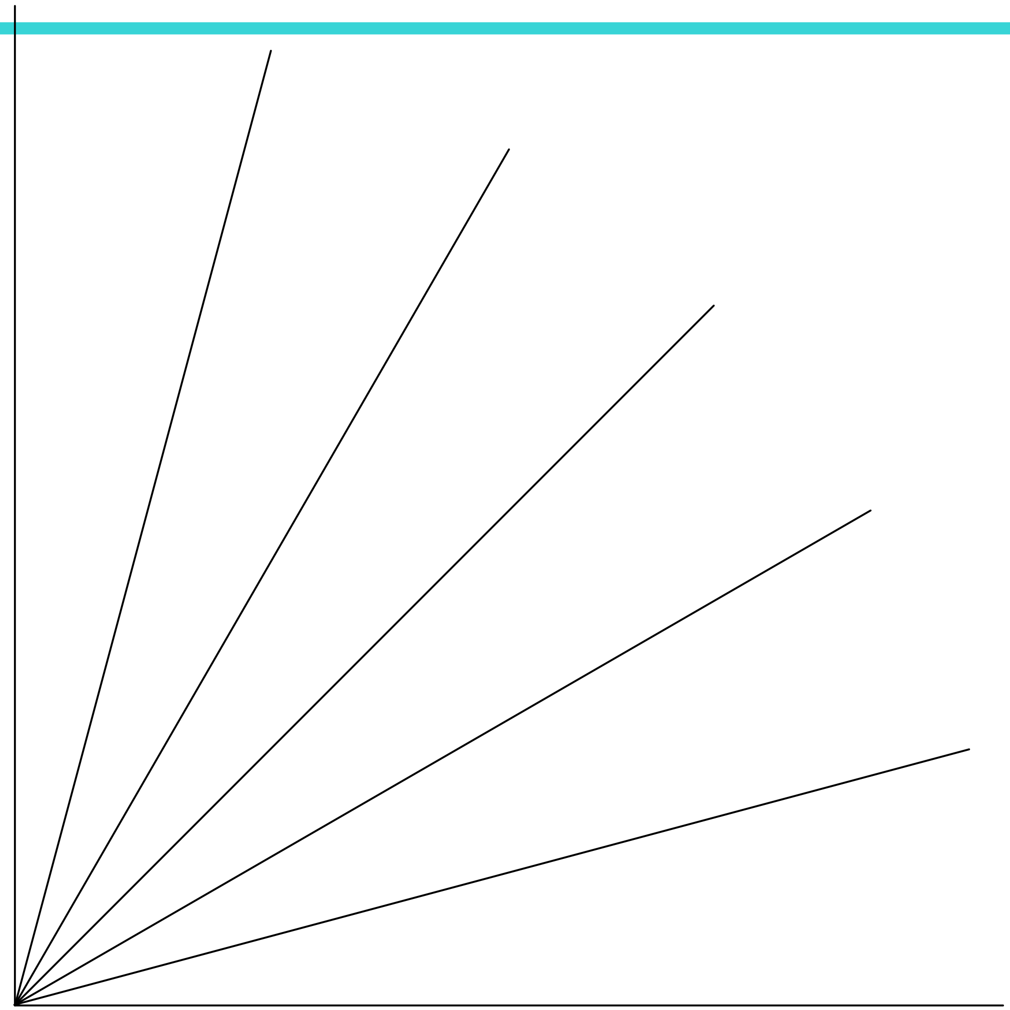
        $y = y + 1$

        $d = d + 2(y_0 - y_1) + 2(x_1 - x_0)$

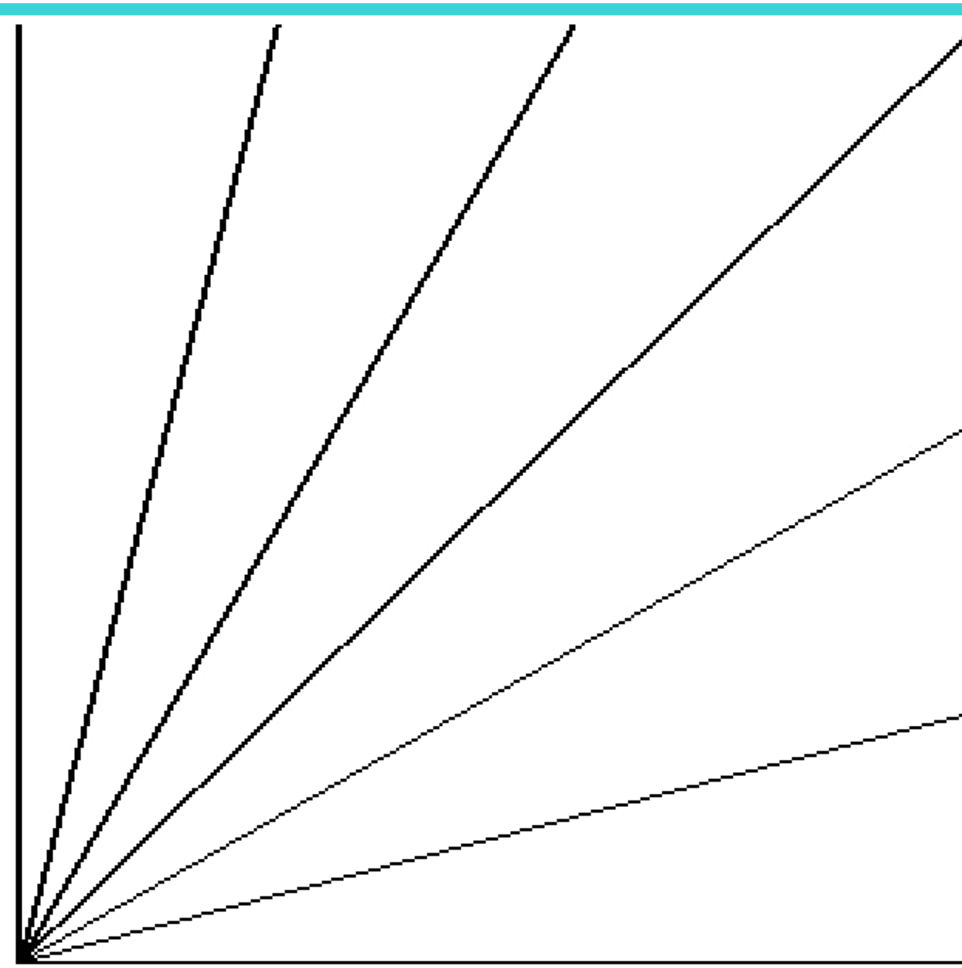
    else

        $d = d + 2(y_0 - y_1)$

Some Lines

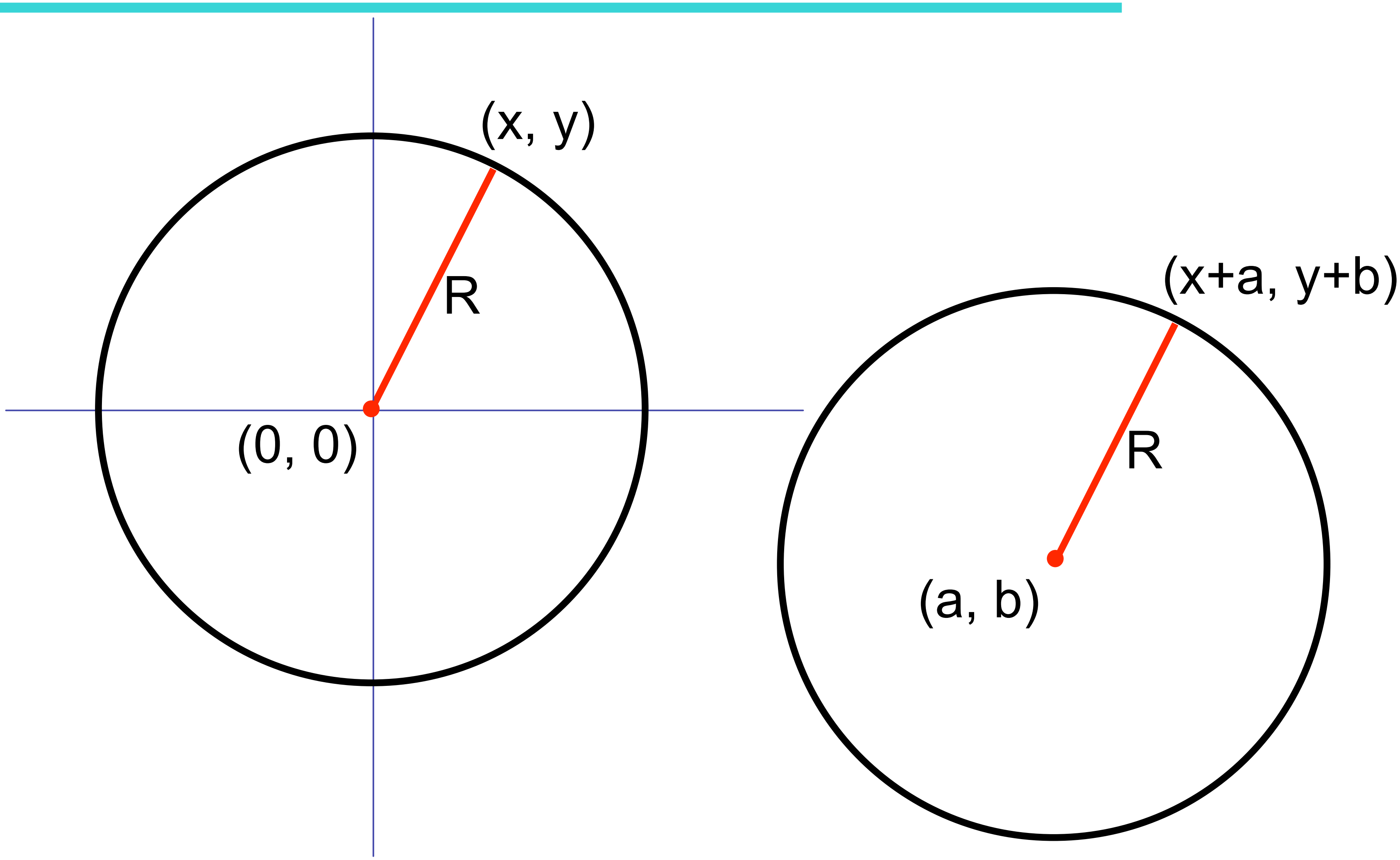


Some Lines Magnified

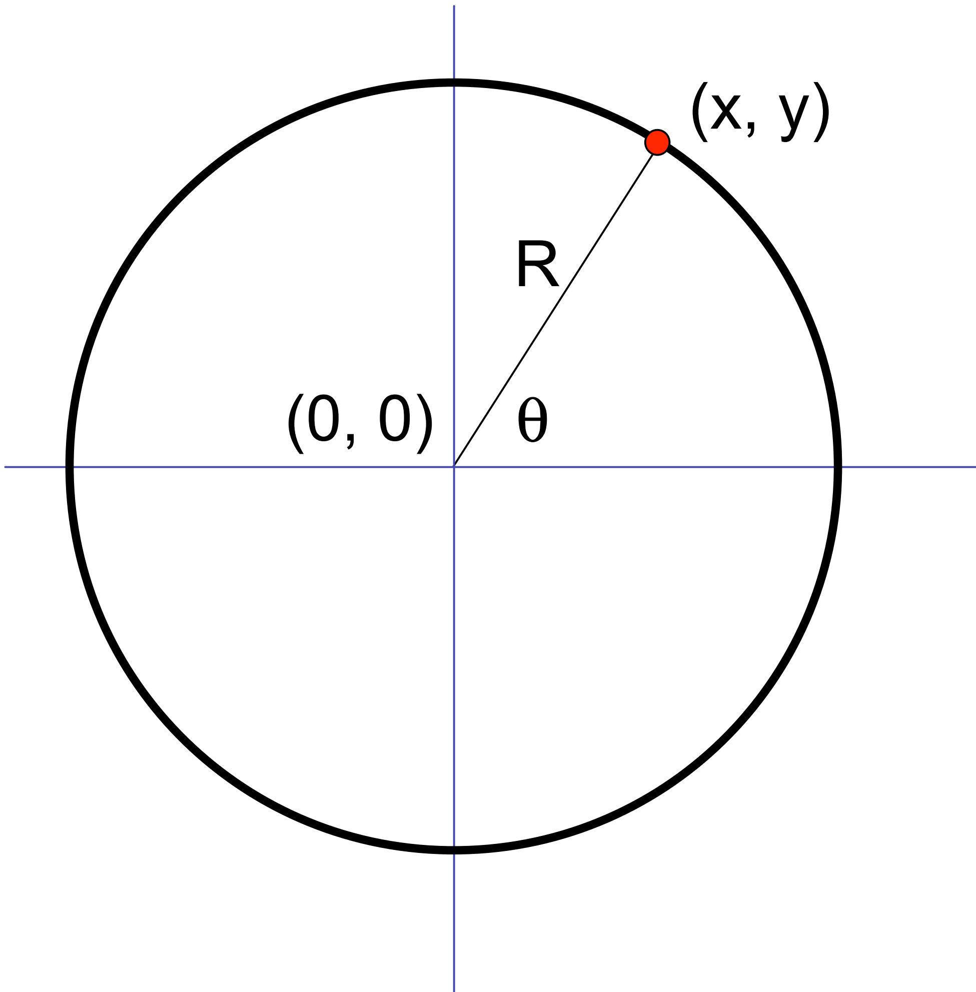


Antialiasing by Downsampling

Circles



Drawing Circles - 1



$x = R\cos(\theta)$

$y = R\sin(\theta)$

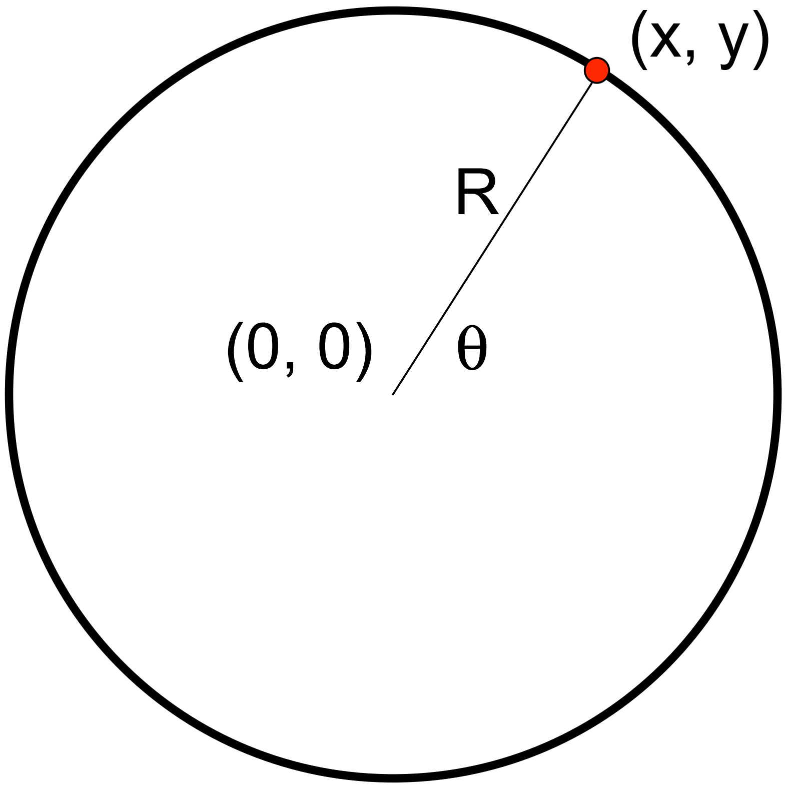
for $\theta = 0$ **to** 360 **do**

  $x = R\cos(\theta)$

  $y = R\sin(\theta)$

  draw(x, y)

Drawing Circles 2



$y^2 + y^2 = R^2$

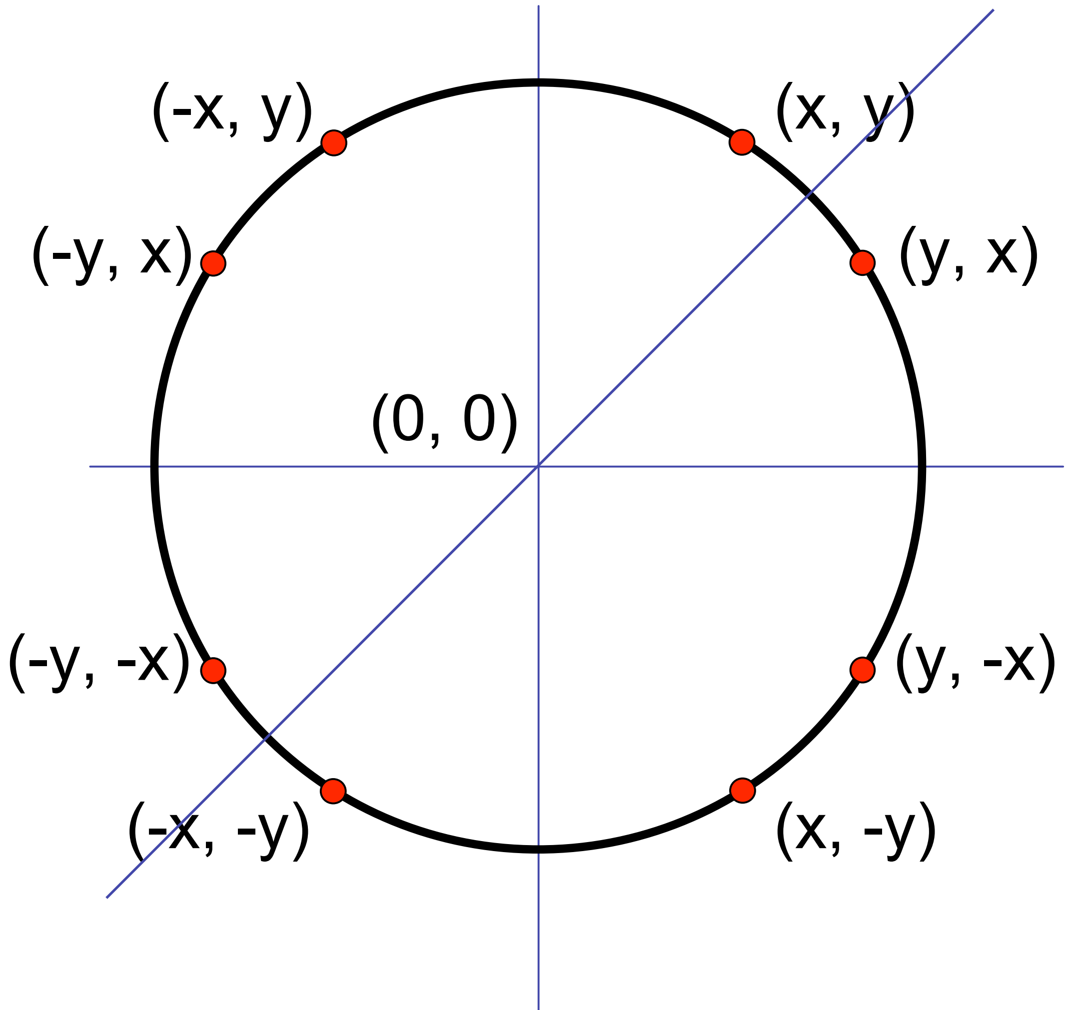
for x = -R **to** R **do**

  y = sqrt(R^2 - x^2)

  draw(x, y)

  draw(x, -y)

Circular Symmetry



Midpoint Circle Algorithm

IN THE TOP OCTANT:

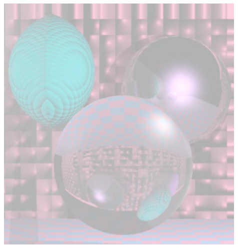
If (x, y) was the last pixel plotted, either

(x + 1, y) or (x + 1, y - 1) will be the next pixel.

Making a Decision Function:

$$d(x, y) = x^2 + y^2 - R^2$$

$$\text{If} \quad \begin{cases} d(x, y) < 0 & (x, y) \text{ is inside the circle.} \\ d(x, y) = 0 & (x, y) \text{ is on the circle.} \\ d(x, y) > 0 & (x, y) \text{ is outside the circle.} \end{cases}$$



Decision Function

Evaluate d at the midpoint of the two possible pixels.

$$d(x + 1, y - \tfrac{1}{2}) = (x + 1)^2 + (y - \tfrac{1}{2})^2 - R^2$$

$$\text{If} \begin{cases} d(x + 1, y - \tfrac{1}{2}) < 0 & \text{midpoint inside circle} & \text{choose } y \\ d(x + 1, y - \tfrac{1}{2}) = 0 & \text{midpoint on circle} & \text{choose } y \\ d(x + 1, y - \tfrac{1}{2}) > 0 & \text{midpoint outside circle} & \text{choose } y - 1 \end{cases}$$

Computing D(x,y) Incrementally

$D(x,y) = d(x + 1, y - ½) = (x + 1)^2 + (y - ½)^2 – R^2$

$D(x + 1,y) – D(x, y)=$
$(x+2)^2 + (y - ½)^2 – R^2 – ((x + 1)^2 + (y - ½)^2 – R^2)$
$=2(x + 1)+ 1$

$D(x + 1,y - 1) – D(x, y)=$
$(x+2)^2 + (y – 3/2)^2 – R^2 – ((x + 1)^2 + (y - ½)^2 – R^2)$
$=2(x+1) + 1 – 2(y – 1)$

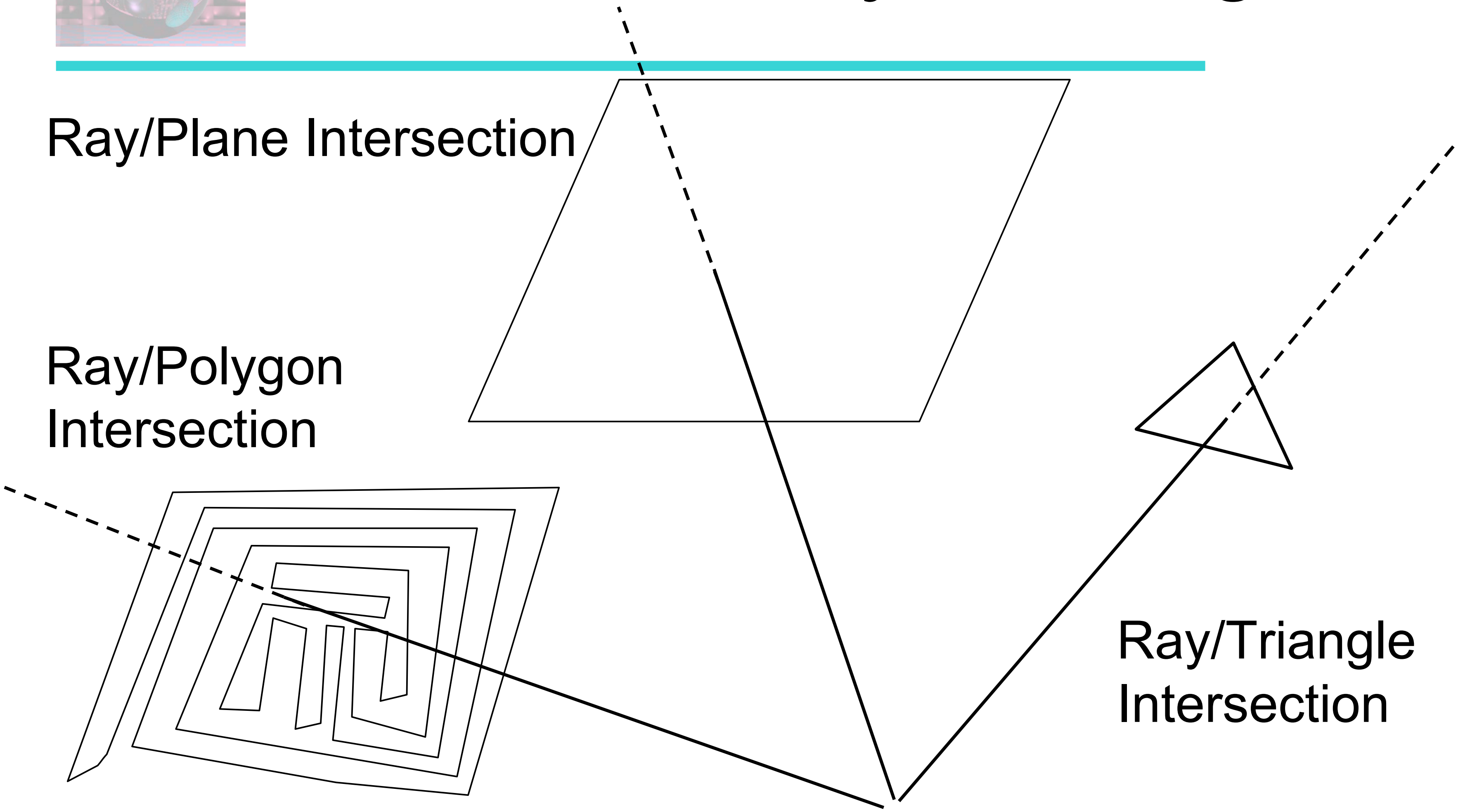
You can also compute the differences incrementally.

Time for a Break

More Ray-Tracing

Ray/Plane Intersection

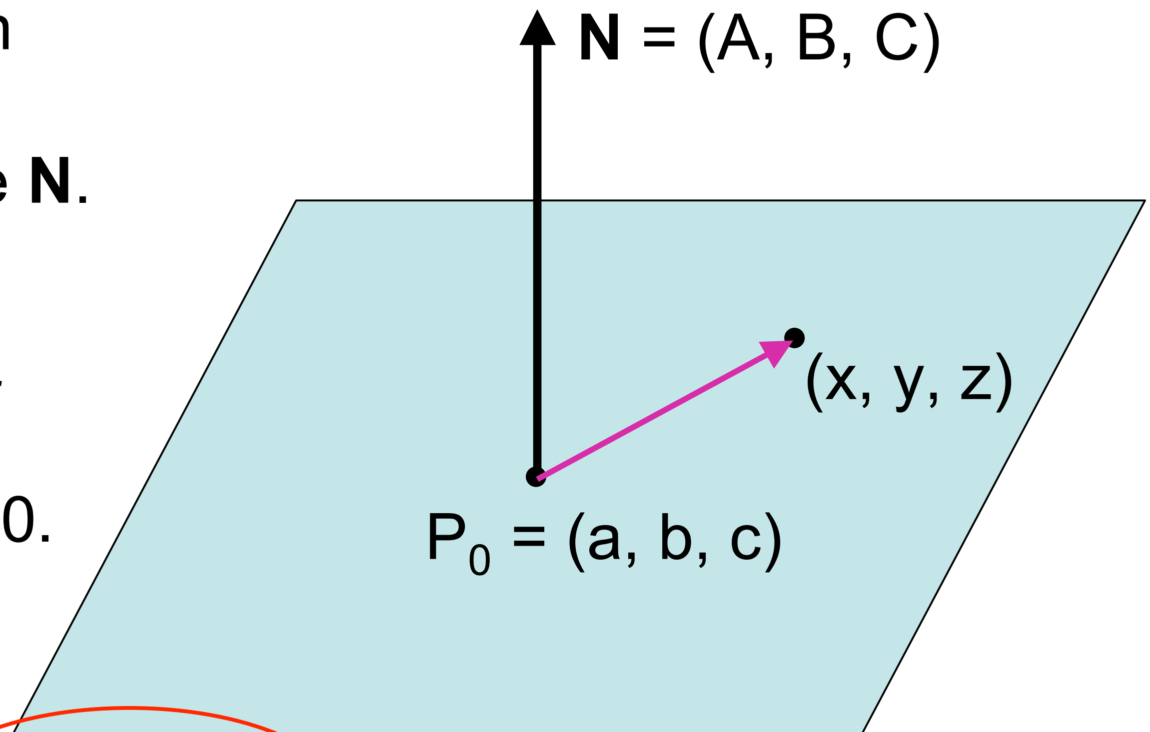
Ray/Polygon
Intersection



Ray/Triangle
Intersection

Equation of a Plane

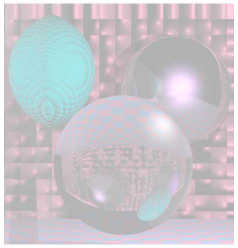
Given a point P_0 on the plane and a normal to the plane **N**.



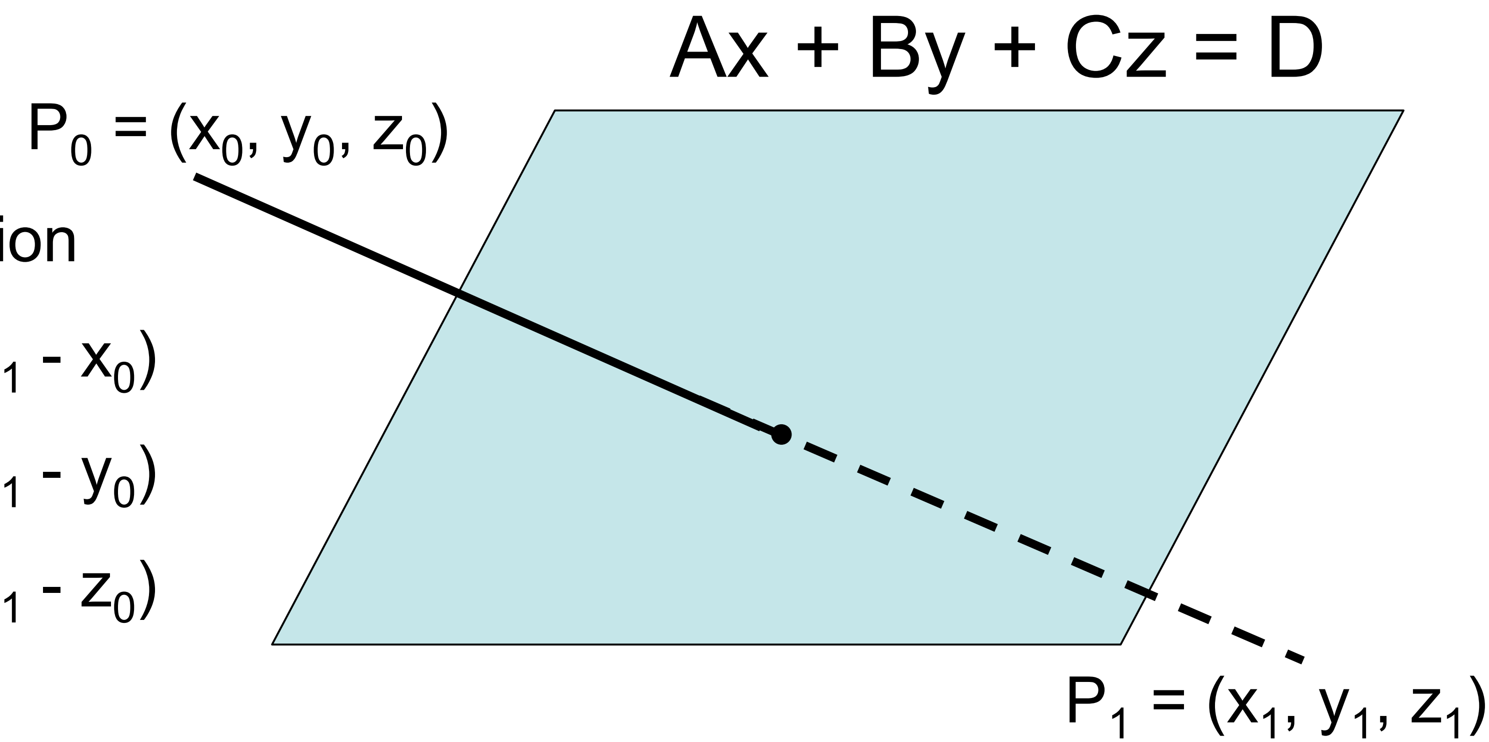
(x, y, z) is on the plane if and only if

$(x-a, y-b, z-c) \cdot \mathbf{N} = 0.$

$$Ax + By + Cz - \underbrace{(Aa + Bb + Cc)}_{D} = 0$$



Ray/Plane Intersection



Ray Equation

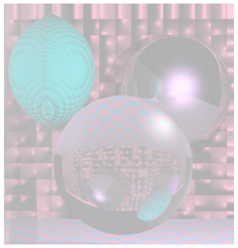
$x = x_0 + t(x_1 - x_0)$

$y = y_0 + t(y_1 - y_0)$

$z = z_0 + t(z_1 - z_0)$

$$A(x_0 + t(x_1 - x_0)) + B(y_0 + t(y_1 - y_0)) + C(z_0 + t(z_1 - z_0)) = D$$

Solve for t. Find x, y, z.



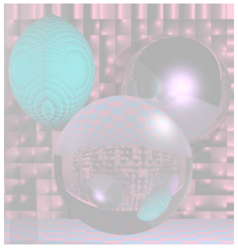
Planes in Your Scenes

- Planes are specified by
  - A, B, C, D or by **N** and P
  - Color and other coefficients are as for spheres
- To search for the nearest object, go through all the spheres and planes and find the smallest t.
- A plane will not be visible if the normal vector (A, B, C) points away from the light.

Ray/Triangle Intersection

Using the Ray/Plane intersection:

- Given the three vertices of the triangle,
  - Find **N**, the normal to the plane containing the triangle.
  - Use **N** and one of the triangle vertices to describe the plane, i.e. Find A, B, C, and D.
  - If the Ray intersects the Plane, find the intersection point and its β and γ.
  - If $0 \leq \beta$ and $0 \leq \gamma$ and $\beta + \gamma \leq 1$, the Ray hits the Triangle.



Ray/Triangle Intersection

Using barycentric coordinates directly: (Shirley pp. 206-208)

Solve

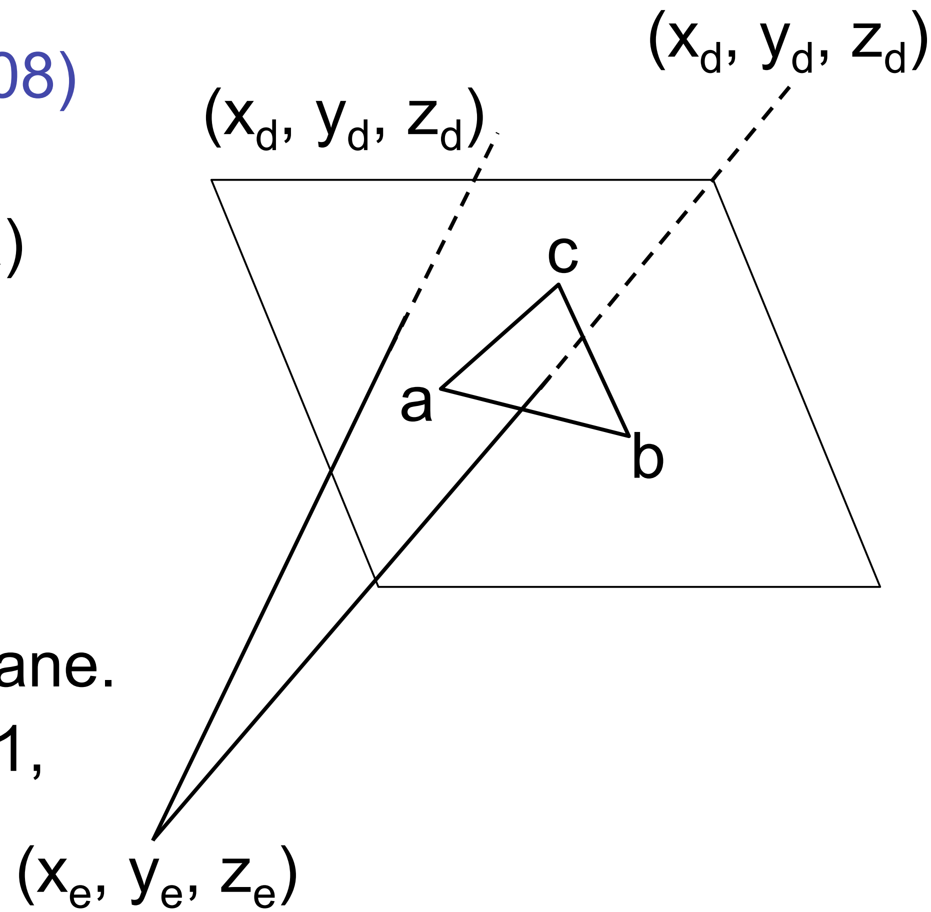
$\mathbf{e} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma\,(\mathbf{c}-\mathbf{a})$

for t, β, and γ.

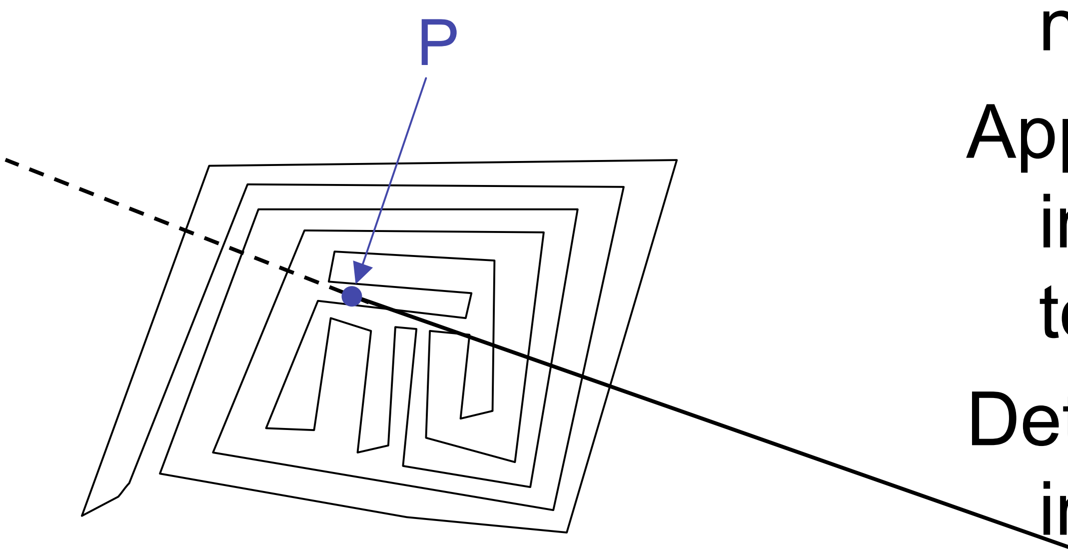
The x, y, and z components give you 3 linear equations in 3 unknowns.

If $0 \le t \le 1$, the Ray hits the Plane.

If $0 \le \beta$ and $0 \le \gamma$ and $\beta + \gamma \le 1$, the Ray hits the Triangle.



Ray/Polygon Intersection

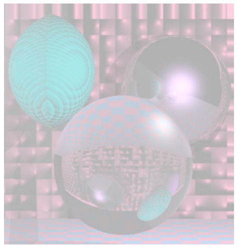


A polygon is given by *n* co-planar points.

Choose 3 points that are not co-linear to find **N**.

Apply Ray/Plane intersection procedure to find P.

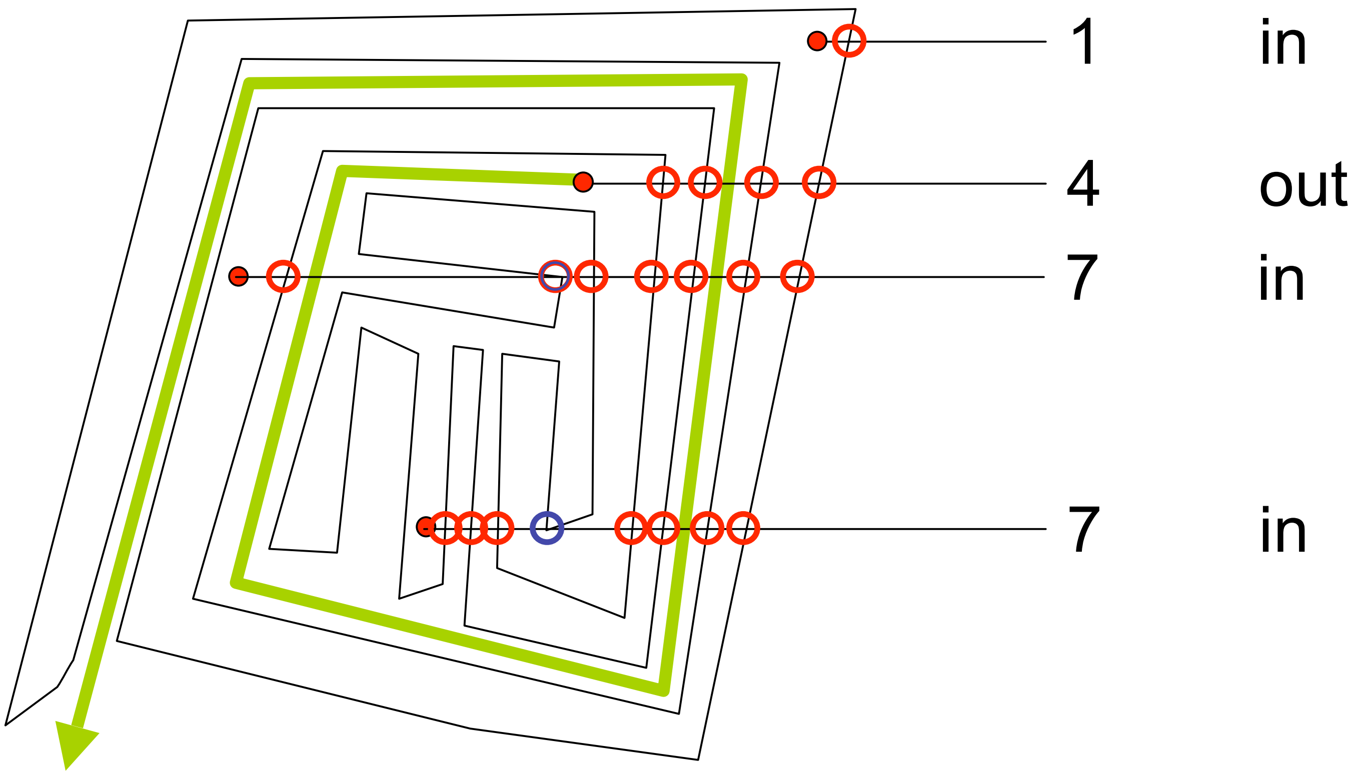
Determine whether P lies inside the polygon.



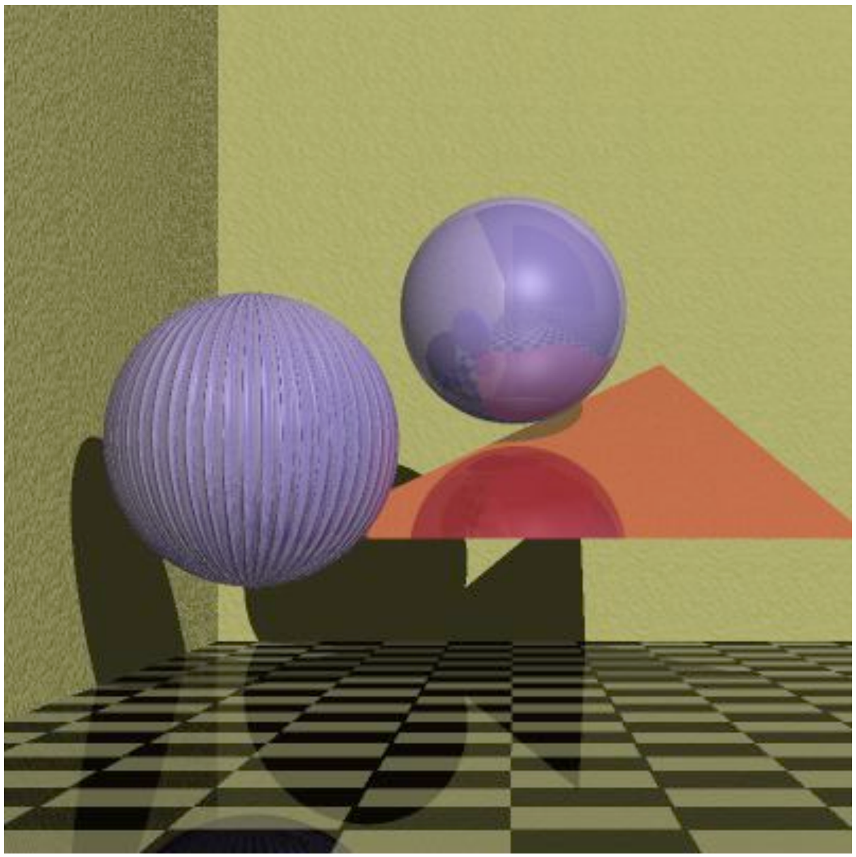
Parity Check

Draw a horizontal half-line from P to the right.

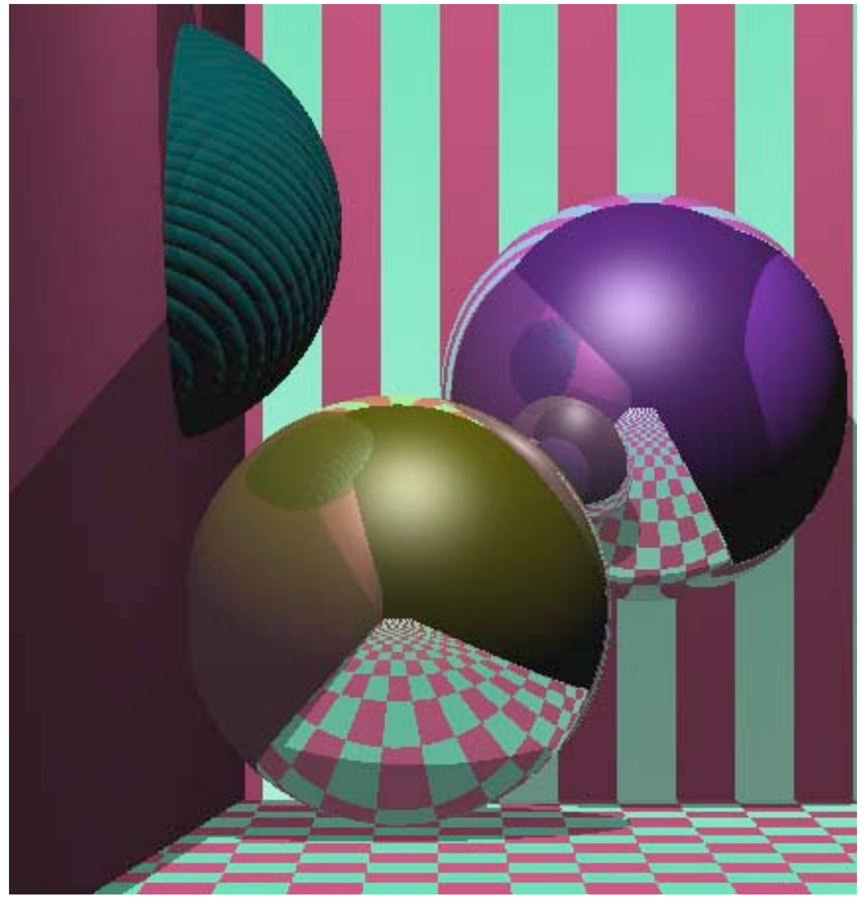
Count the number of times the half-line crosses an edge.



Images with Planes and Polygons



Images with Planes and Polygons



Scan Line Polygon Fill

Polygon Data Structure

edges

xmin	ymax	1/m	• →

1	6	8/4	• →

(9, 6)

(1, 2)

xmin = x value at lowest y

ymax = highest y

Polygon Data Structure

Edge Table (ET) has a list of edges for each scan line.

13

12

11

10 $\rightarrow$ e6

9

8

7 $\rightarrow$ e4 $\rightarrow$ e5

6 $\rightarrow$ e3 $\rightarrow$ e7 $\rightarrow$ e8

5

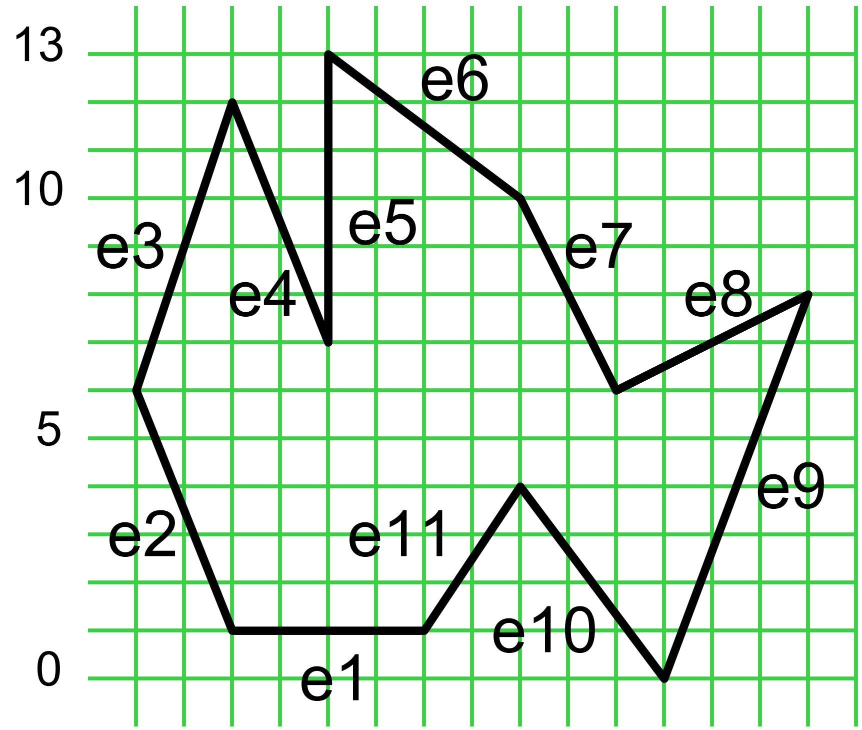
4

3

2

1 $\rightarrow$ e2 $\rightarrow$ e1 $\rightarrow$ e11

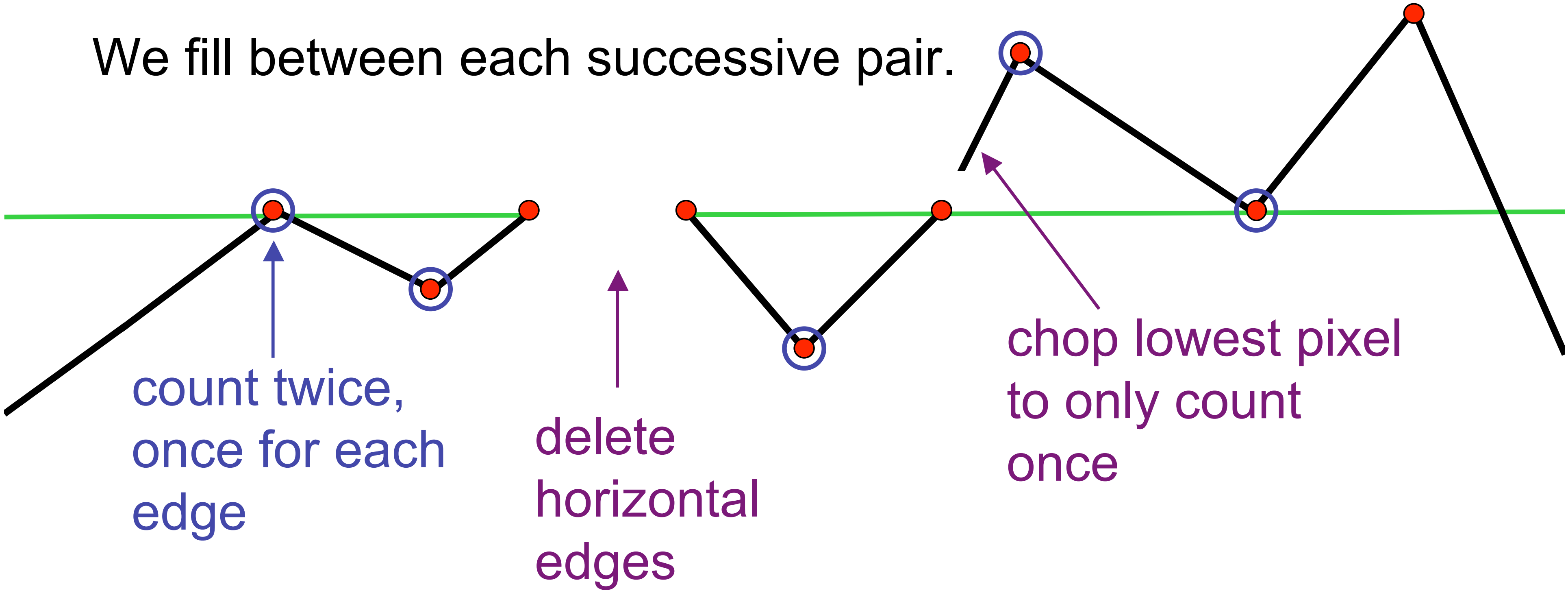
0 $\rightarrow$ e10 $\rightarrow$ e9



Preprocessing the edges

For a closed polygon, there should be an even number of crossings at each scan line.

We fill between each successive pair.

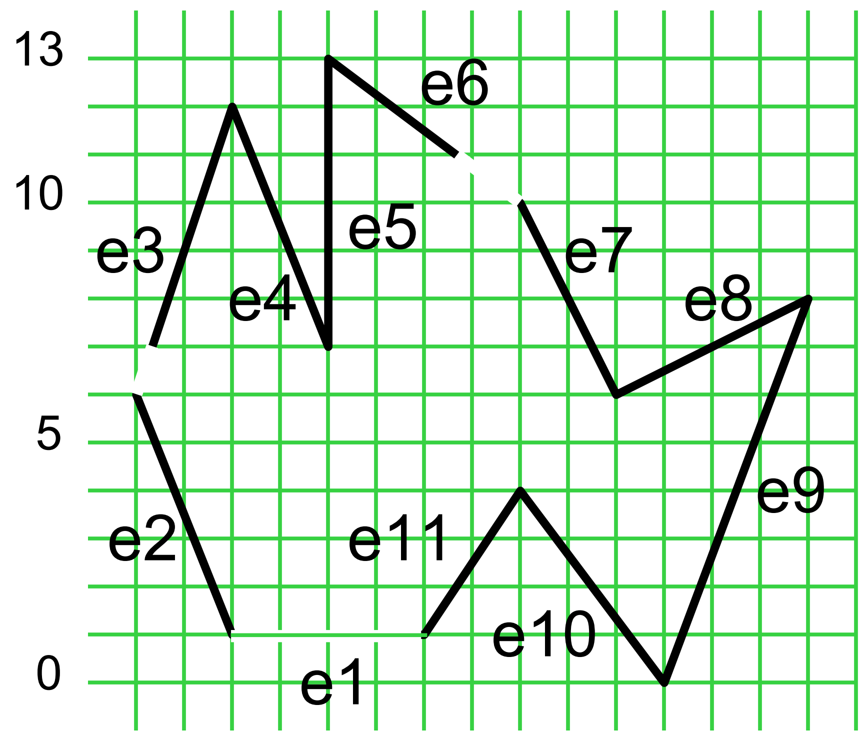


Polygon Data Structure

after preprocessing

Edge Table (ET) has a list of edges for each scan line.

13	
12	
11	→ e6
10	
9	
8	
7	→ e3 → e4 → e5
6	→ e7 → e8
5	
4	
3	
2	
1	→ e2 → e11
0	→ e10 → e9



The Algorithm

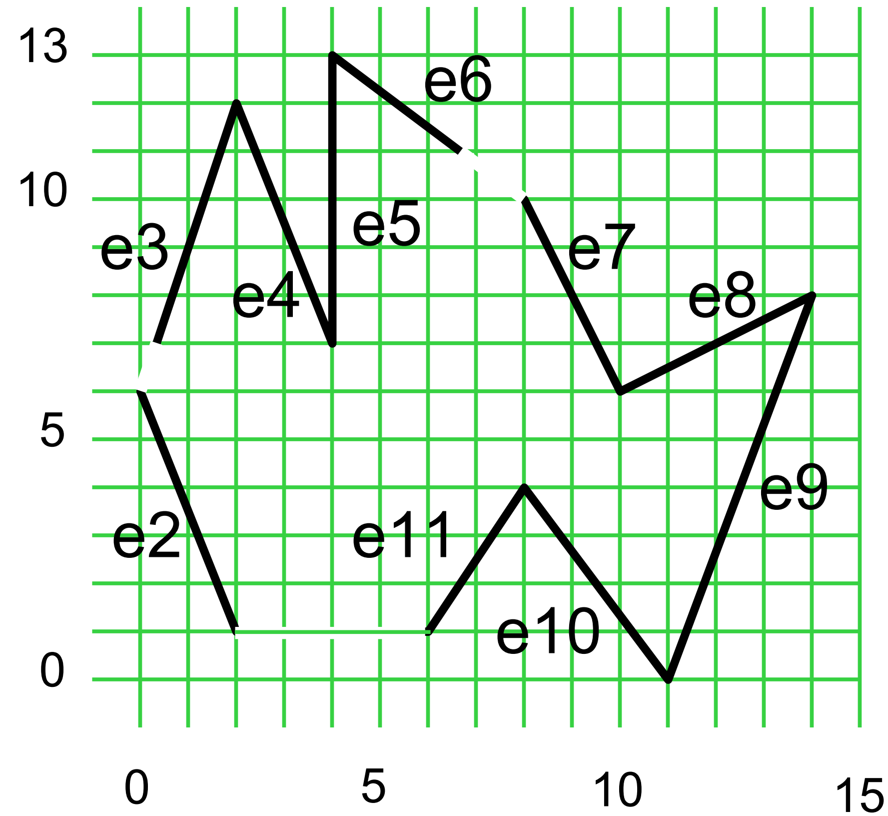
1. Start with smallest nonempty y value in ET.
2. Initialize SLB (Scan Line Bucket) to *nil*.
3. While current y ≤ top y value:
   a. Merge y bucket from ET into SLB; sort on xmin.
   b. Fill pixels between rounded pairs of x values in SLB.
   c. Remove edges from SLB whose ytop = current y.
   d. Increment xmin by 1/m for edges in SLB.
   e. Increment y by 1.

Running the Algorithm

ET

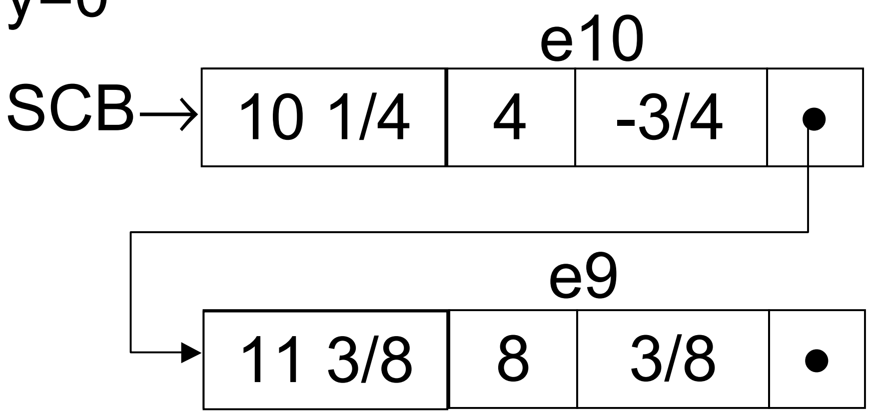
13	
12	
11	$\rightarrow$ e6
10	
9	
8	
7	$\rightarrow$ e3 $\rightarrow$ e4 $\rightarrow$ e5
6	$\rightarrow$ e7 $\rightarrow$ e8
5	
4	
3	
2	
1	$\rightarrow$ e2 $\rightarrow$ e11
0	$\rightarrow$ e10 $\rightarrow$ e9

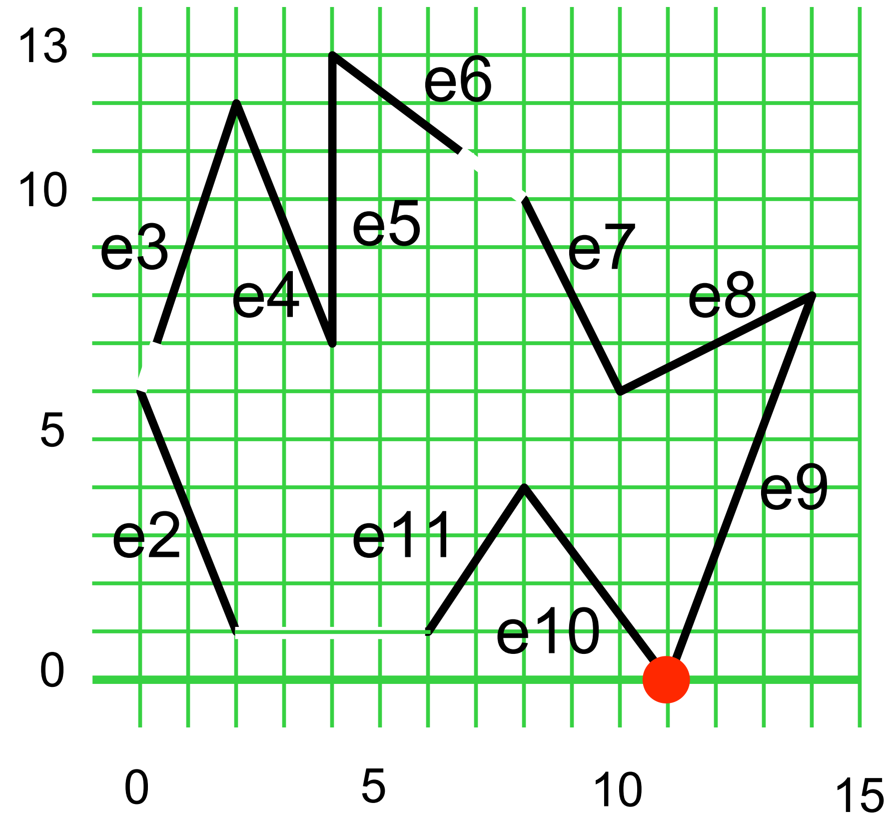
	xmin	ymax	1/m
e2	2	6	-2/5
e3	1/3	12	1/3
e4	4	12	-2/5
e5	4	13	0
e6	6 2/3	13	-4/3
e7	10	10	-1/2
e8	10	8	2
e9	11	8	3/8
e10	11	4	-3/4
e11	6	4	2/3



Running the Algorithm

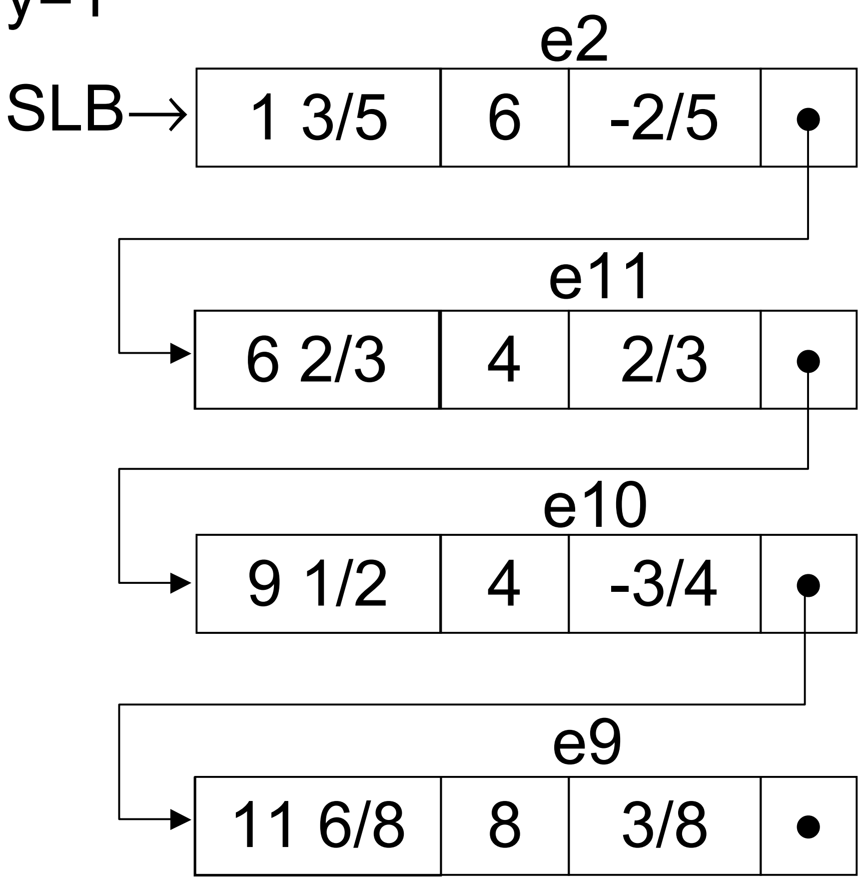
y=0

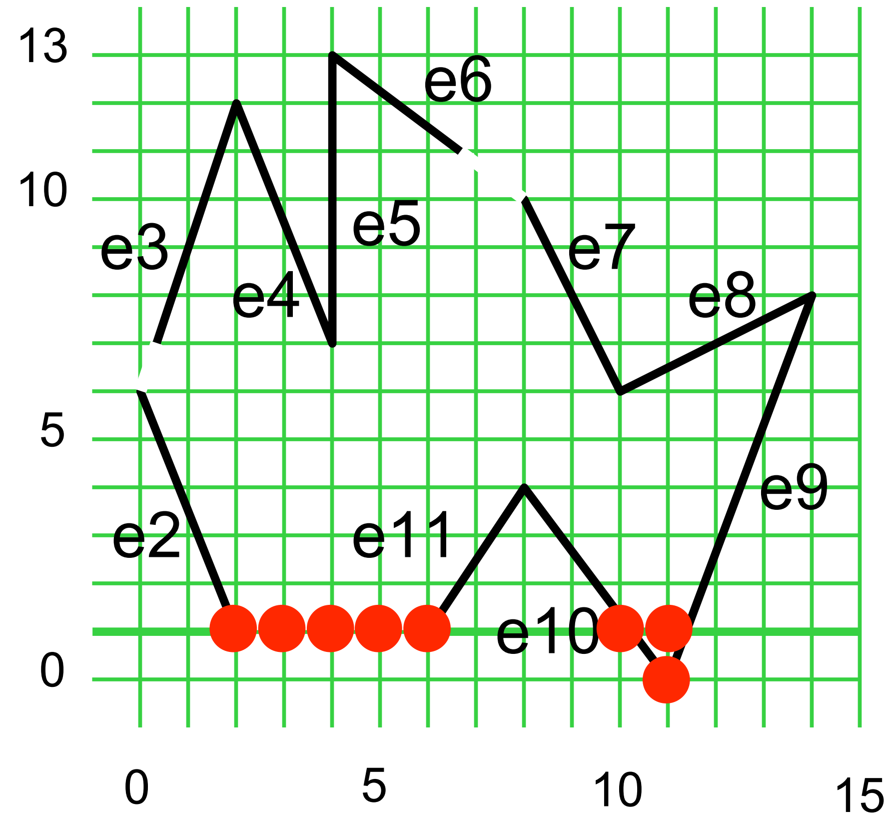


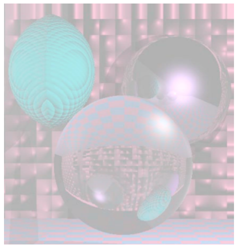


Running the Algorithm

y=1







Running the Algorithm

y=2

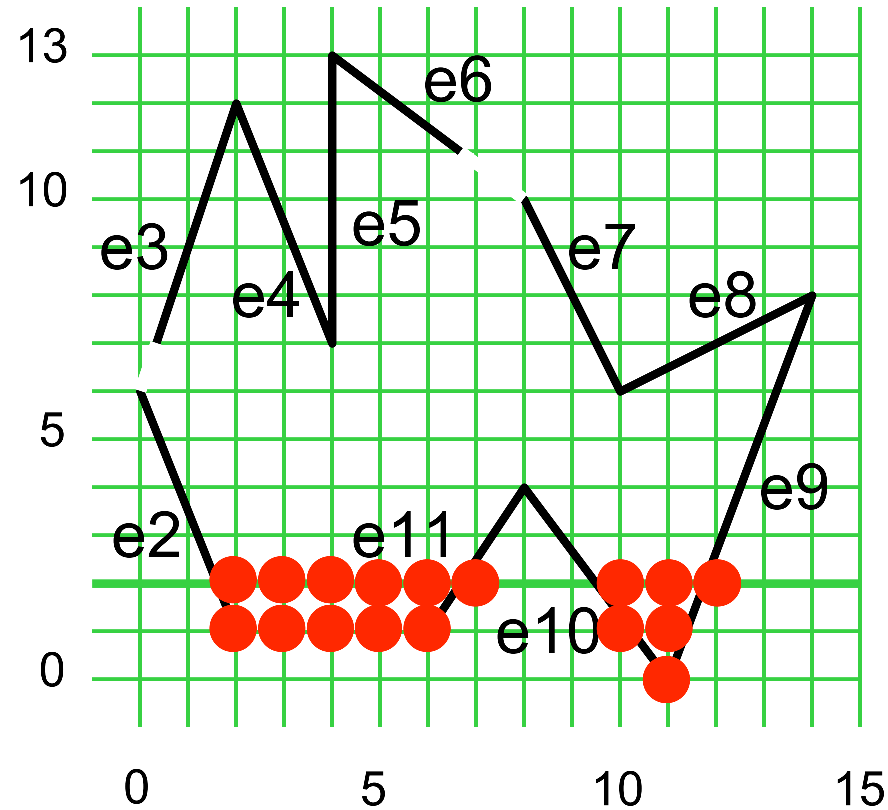
SLB→

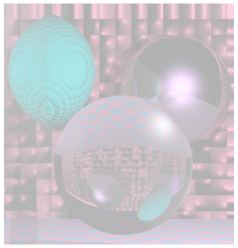
	e2		
1 1/5	6	-2/5	•

	e11		
7 1/3	4	2/3	•

	e10		
8 3/4	4	-3/4	•

	e9		
12 1/8	8	3/8	•



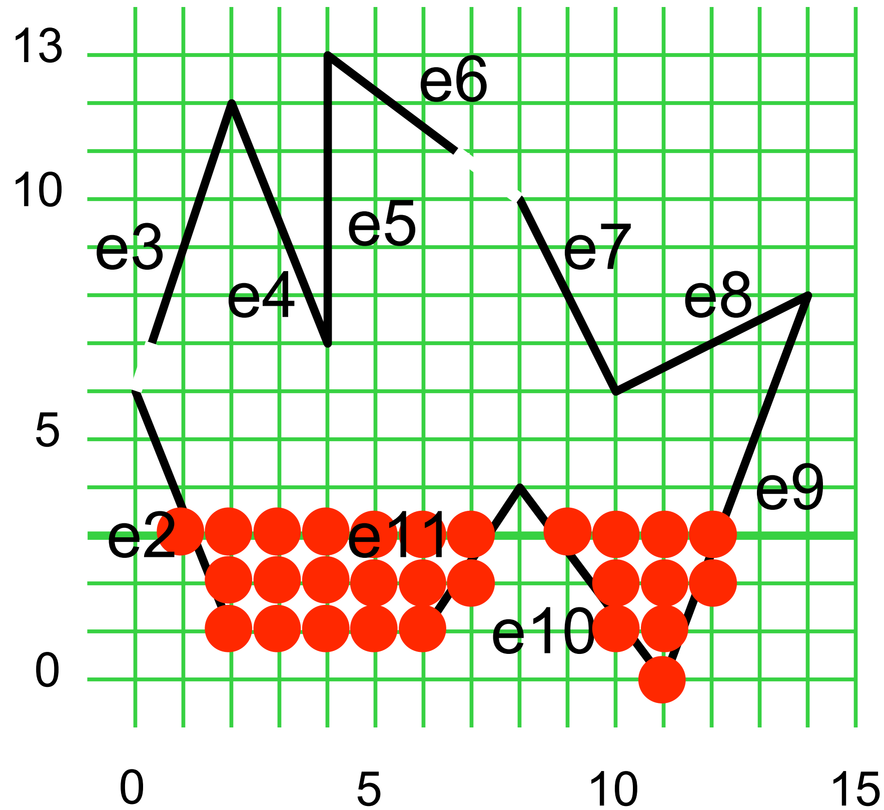


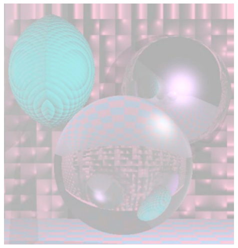
Running the Algorithm

y=3

SLB→

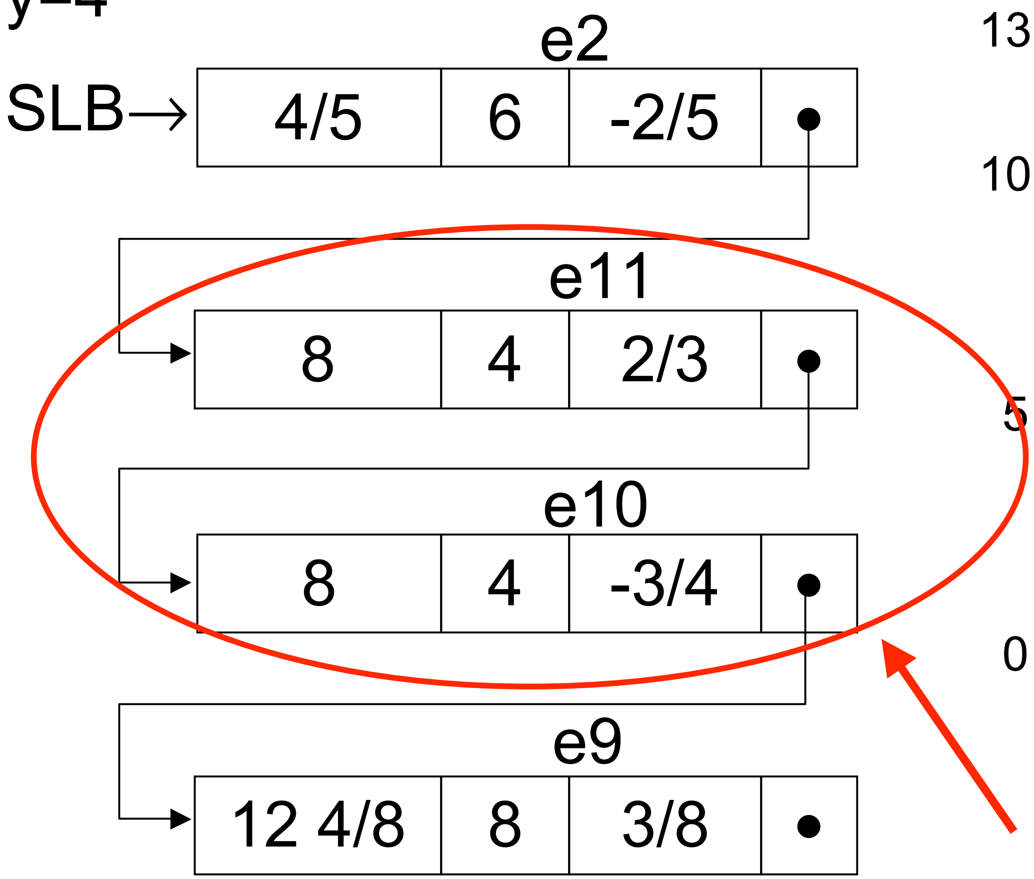
e2	4/5	6	-2/5	•
e11	8	4	2/3	•
e10	8	4	-3/4	•
e9	12 4/8	8	3/8	•

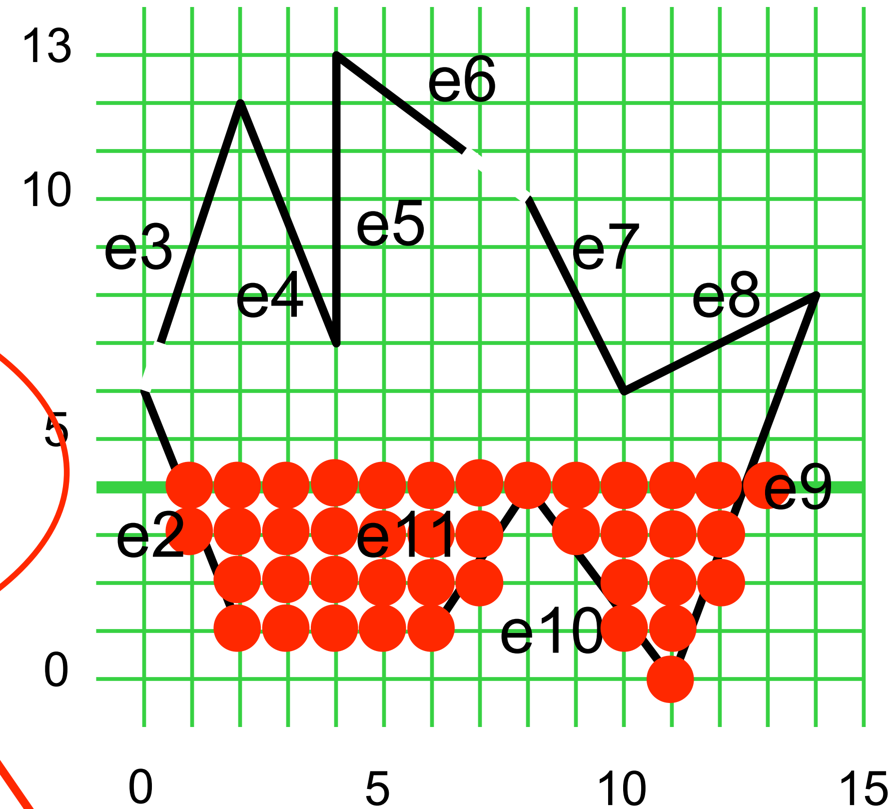




Running the Algorithm

y=4

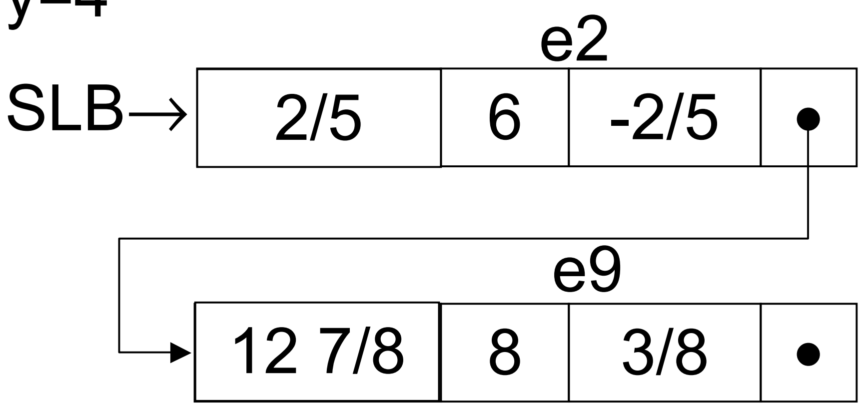




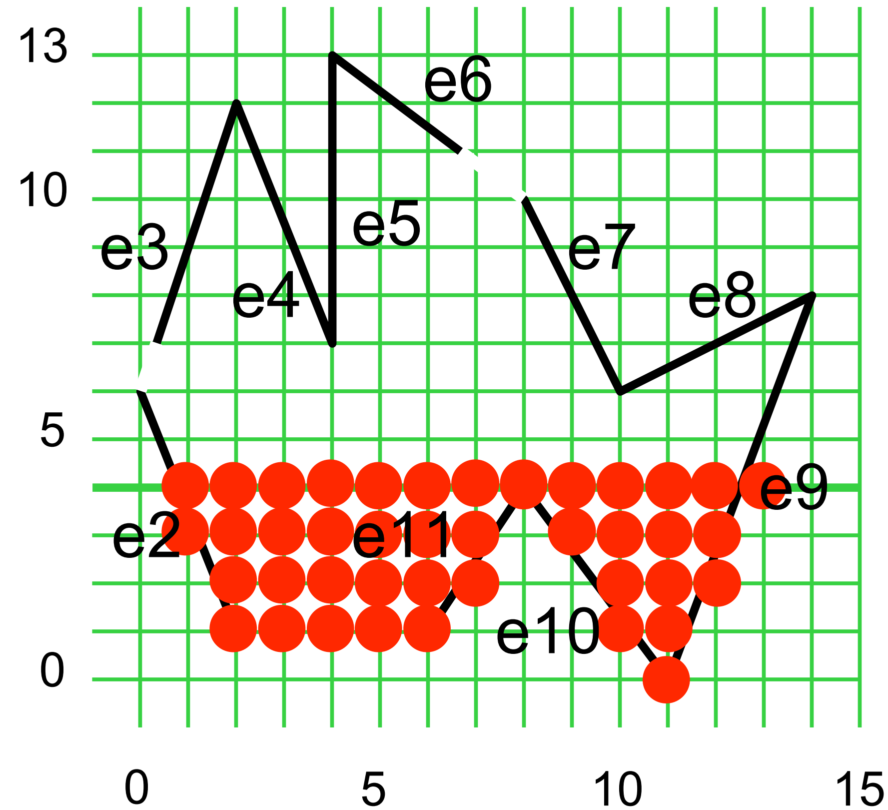
Remove these edges.

Running the Algorithm

y=4



e11 and e10 are removed.



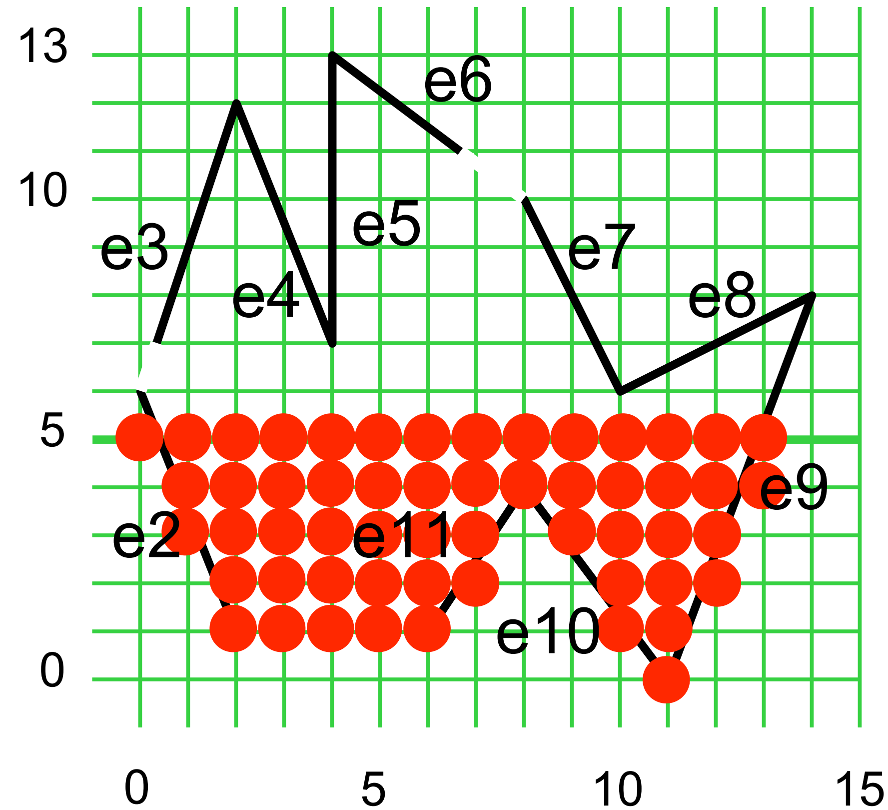
Running the Algorithm

y=5

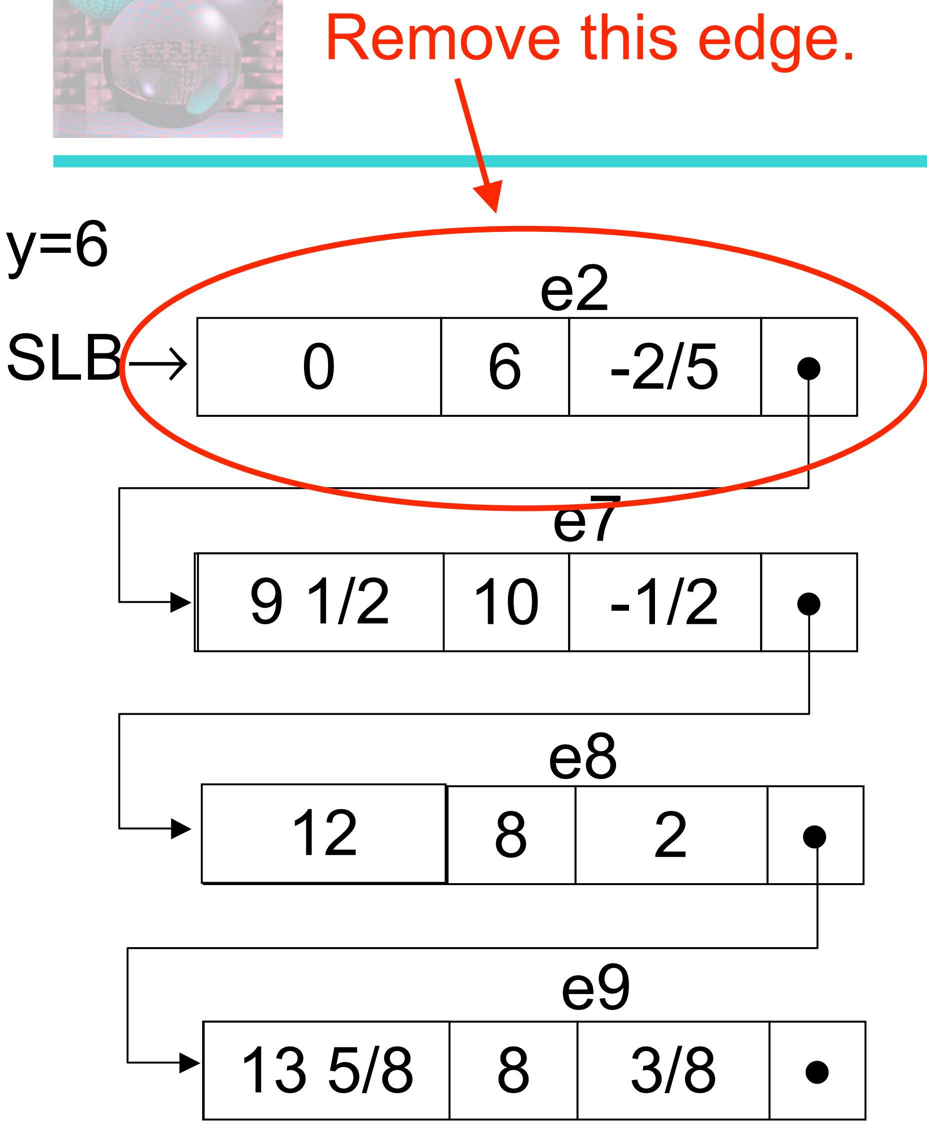
SLB→

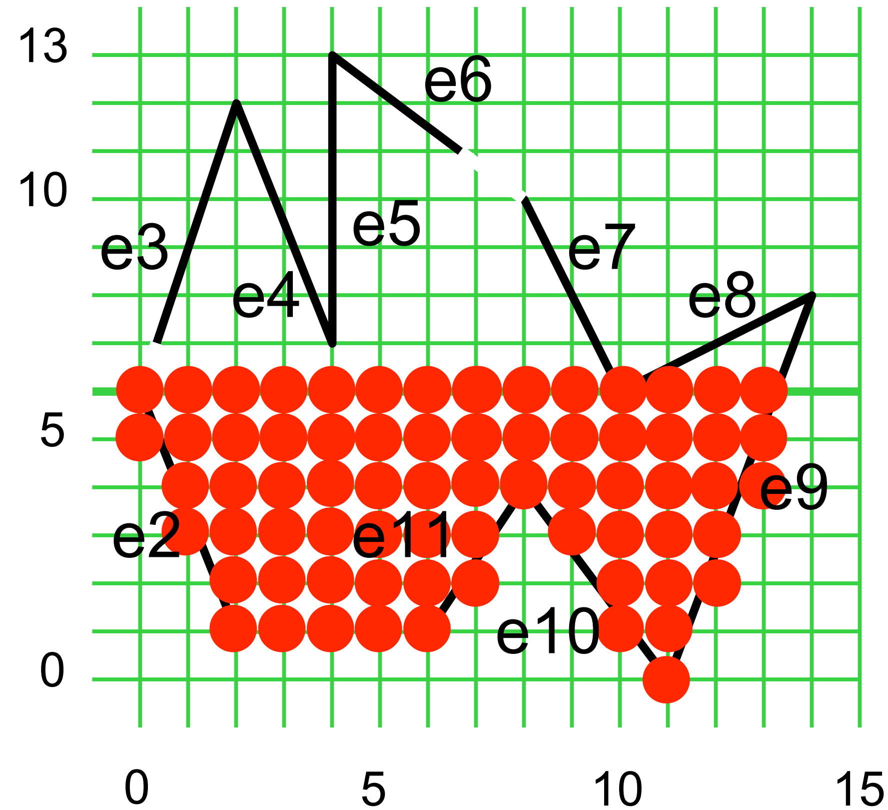
e2			
0	6	-2/5	•

e9			
13 2/8	8	3/8	•



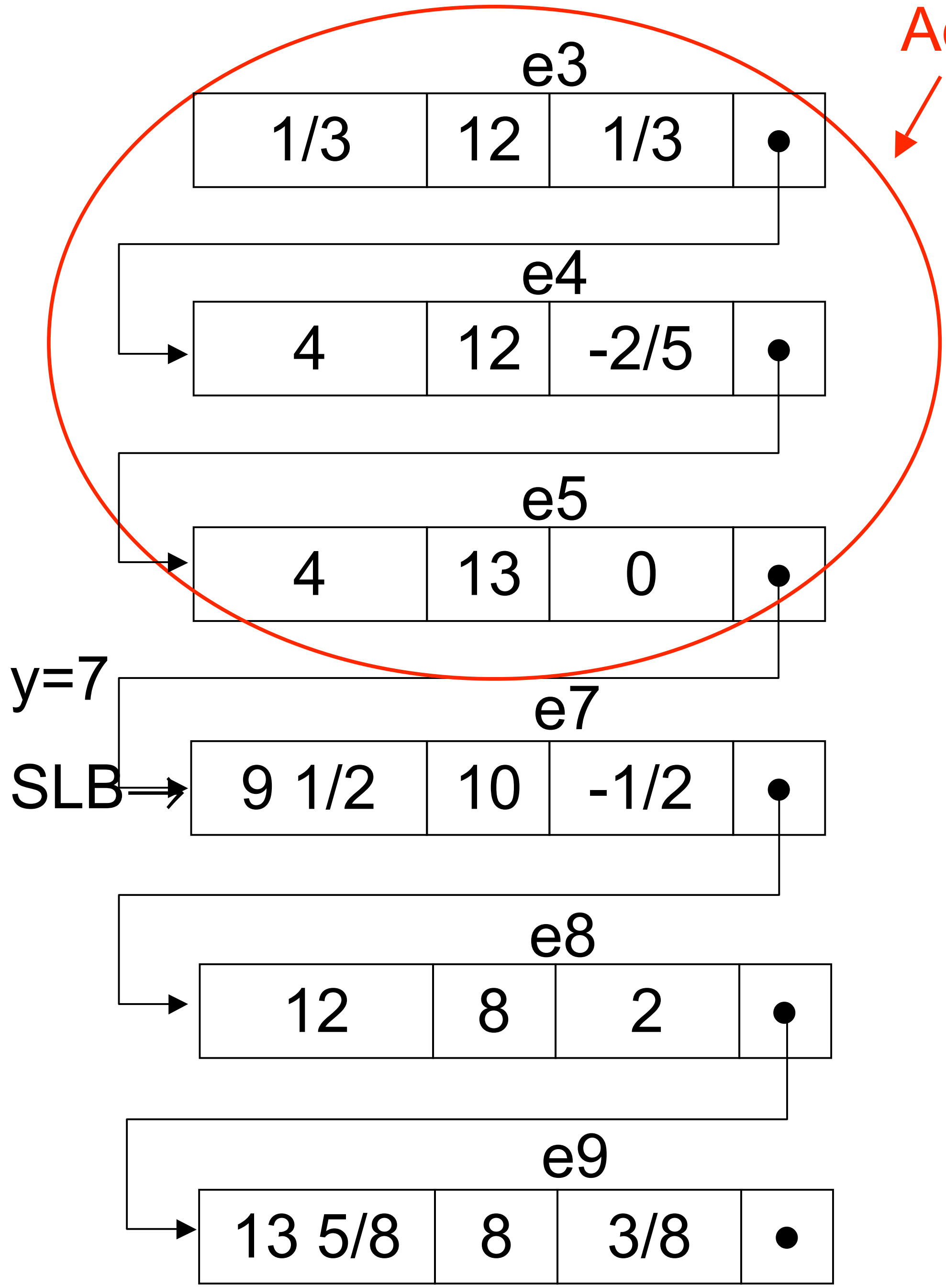
Running the Algorithm

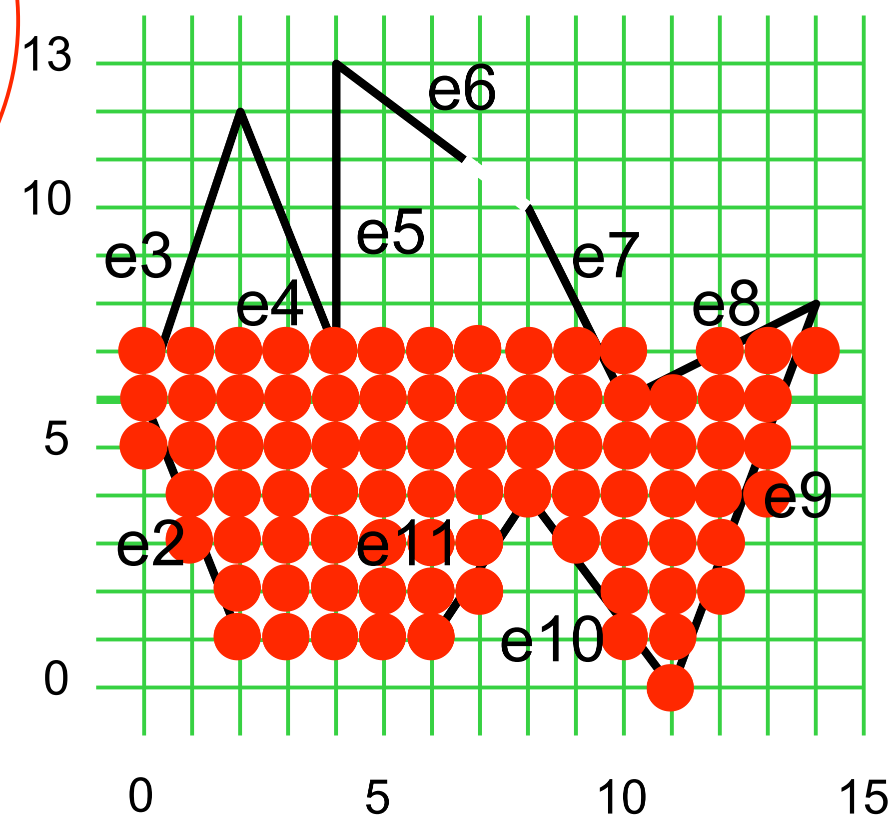




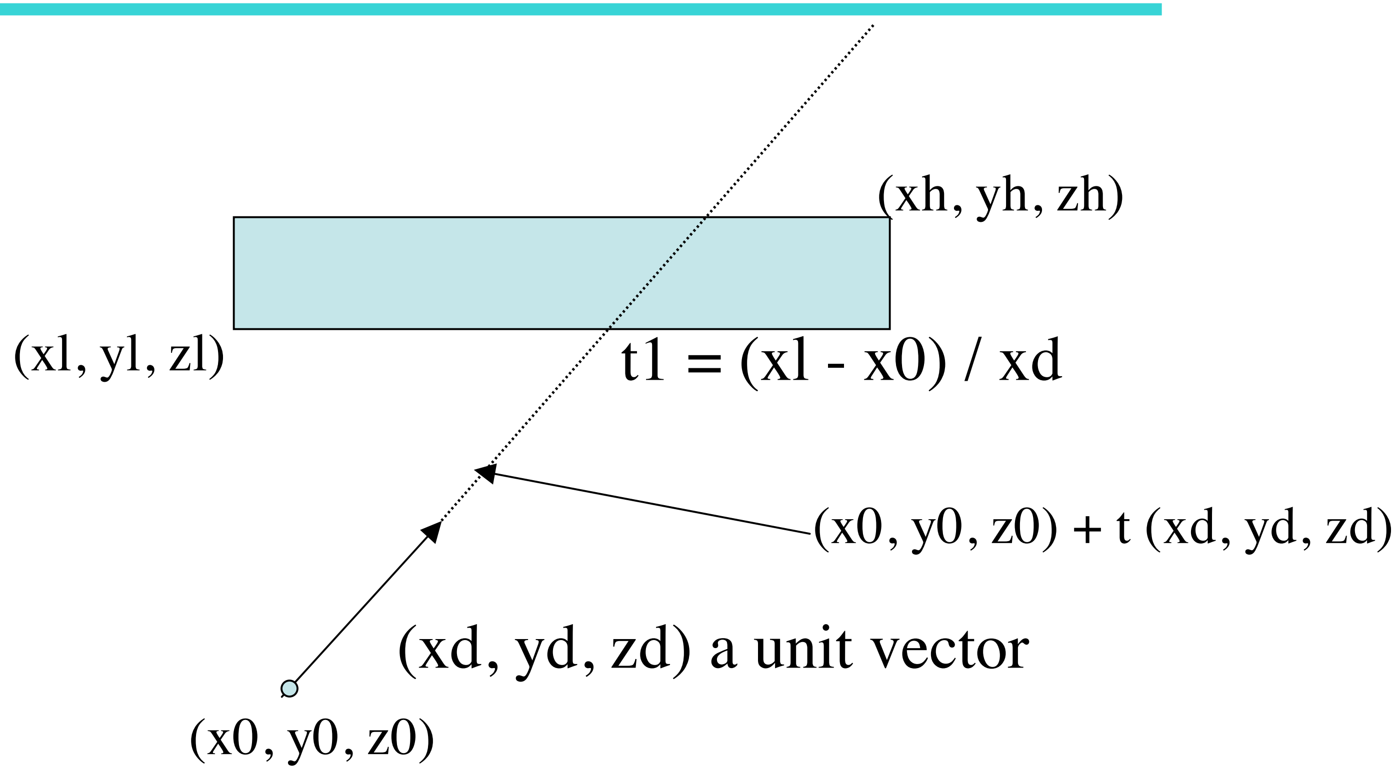
Running the Algorithm

Add these edges.





Ray Box Intersection



Ray Box Intersection

http://courses.csusm.edu/cs697exz/ray_box.htm

or see Watt pages 21-22

Box: minimum extent Bl = (xl, yl, zl) maximum extent Bh = (xh, yh, zh)
Ray: R0 = (x0, y0, z0) , Rd= (xd, yd, zd) ray is R0 + tRd

Algorithm:

1. Set tnear = -INFINITY , tfar = +INFINITY
2. For the pair of X planes
    1. if zd = 0, the ray is parallel to the planes so:
        - if x0 < x1 or x0 > xh return FALSE (origin not between planes)
    2. else the ray is not parallel to the planes, so calculate intersection distances of planes
        - t1 = (xl - x0) / xd (time at which ray intersects minimum X plane)
        - t2 = (xh - x0) / xd (time at which ray intersects maximum X plane)
        - if t1 > t2 , swap t1 and t2
        - if t1 > tnear , set tnear = t1
        - if t2 < tfar, set tfar= t2
        - if tnear > tfar, box is missed so return FALSE
        - if tfar < 0 , box is behind ray so return FALSE
3. Repeat step 2 for Y, then Z
4. All tests were survived, so return TRUE