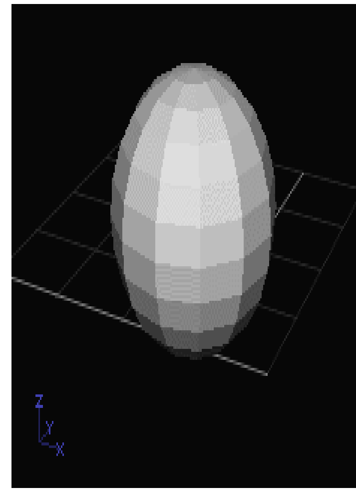
Chapter 3

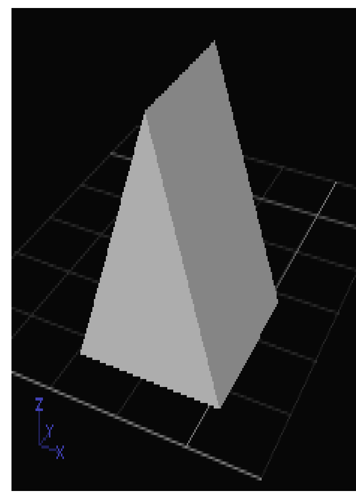
Display Primitives

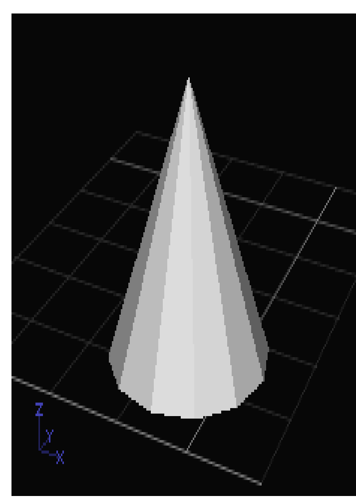
Saminda Premaratne

Definitions

- **CG API** – Computer Graphics Application Programming Interface (OpenGL, DirectX)
- **Graphics primitives** – functions in the API that describe picture components
- How could we describe an object?
  - Typically focus on ***object shape***
  - Define an object's shape with **geometric primitives**
  - Span of primitives are defined by the API
  - What are some types?
    - Lines, Triangles, Quadrics, Conic sections, Curved surfaces

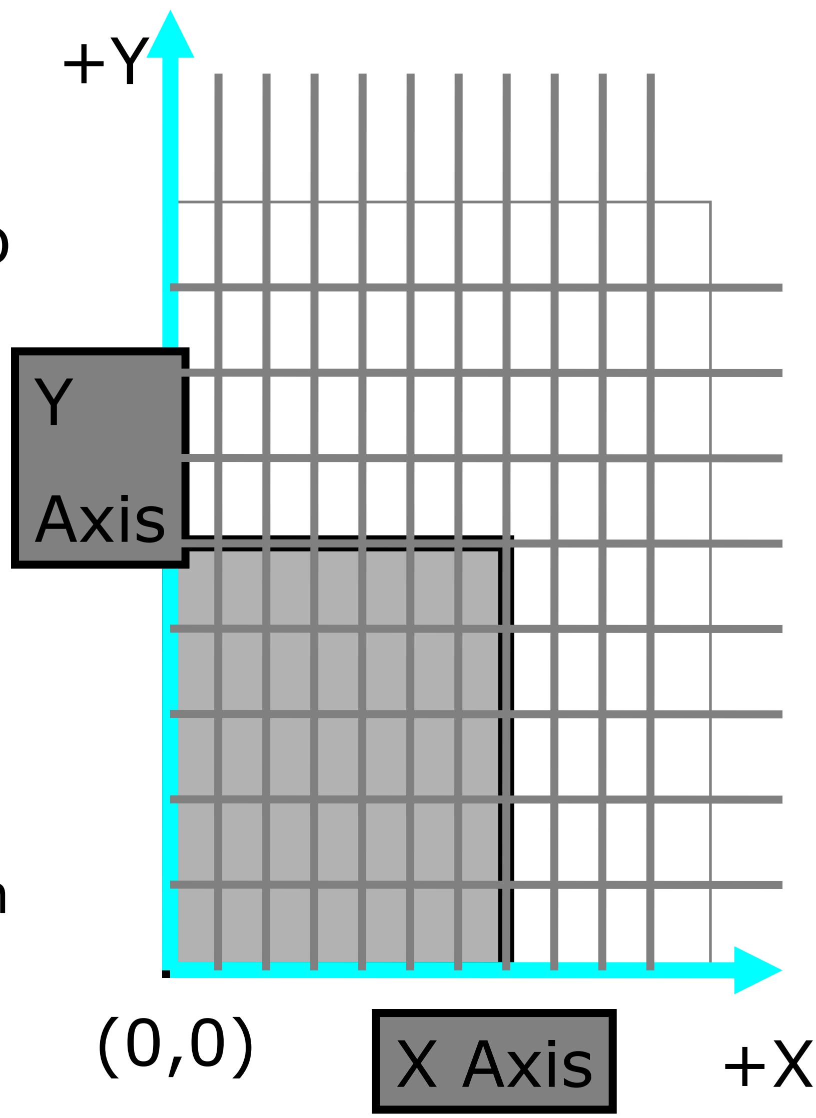






Two Dimensional Images

- Use Cartesian coordinates
- We label the two axes as
  - X (horizontal)
  - Y (vertical)
- Origin is in the lower left
- How big is the space?
  - So what is the image we see on a screen?
  - We call this space the world coordinate system



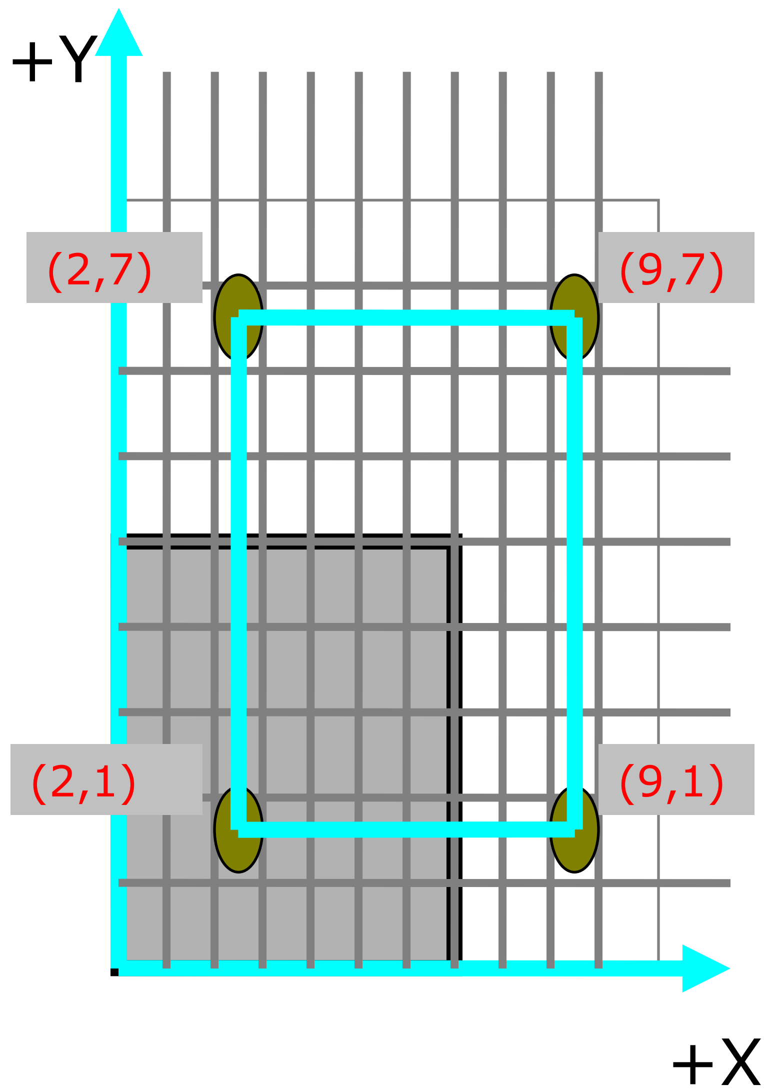
Partition the space into pixels

1. Define a set of points (vertices) in 2D space.

2. Given a set of vertices, draw lines between consecutive vertices.

3. If you were writing OpenGL yourself, let's talk about low level calls

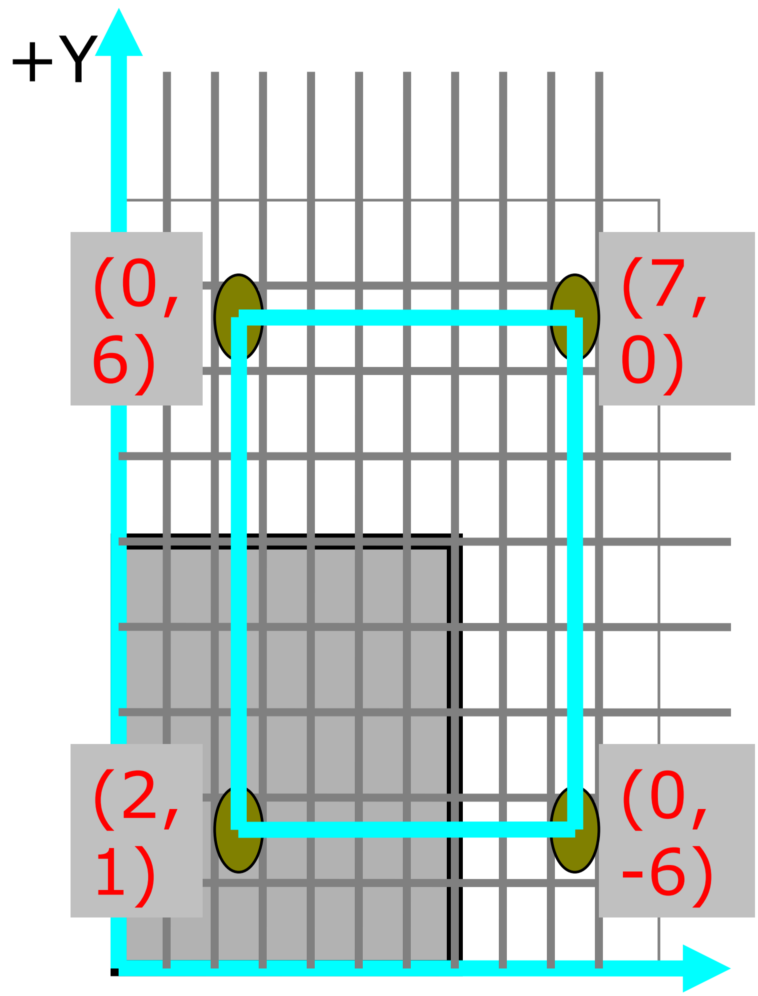
4. What about 2D vs 3D?



Screen Coordinates – references to frame buffer locations

Absolute and Relative Coordinate Specifications

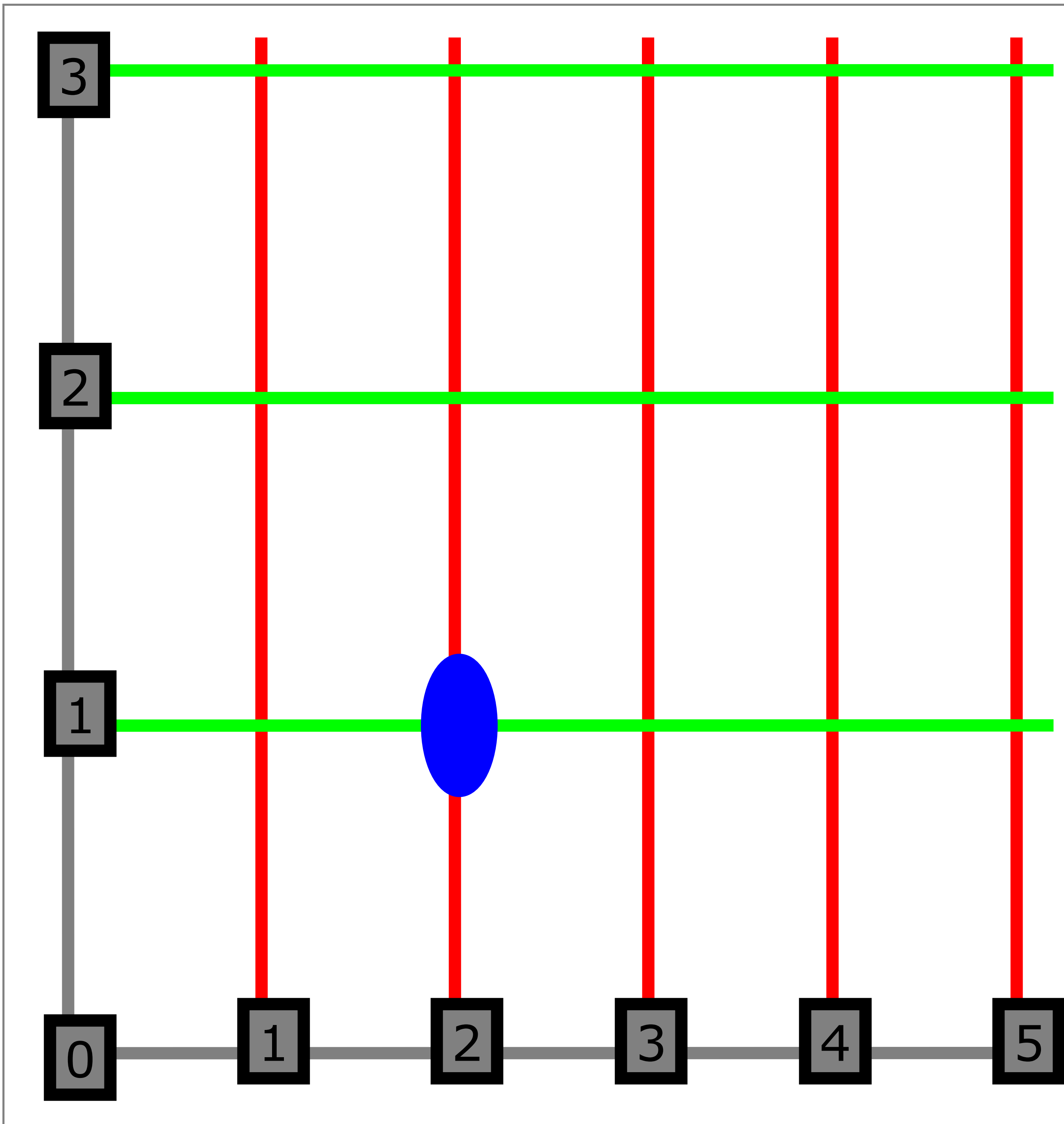
- **Absolute coordinates** – location specified as a relationship to the origin
- **Relative coordinates** – location specified as a relationship to other points
  - Good for pen/plotters
  - Publishing/layou t
  - Allows for a very object oriented approach



What is a “pixel”

Q: What is a pixel? A square or a point?

Q: Where is (2,1)?



Scan Converting Lines

Line Drawing

- Draw a line on a raster screen between two points
- What’s wrong with statement of problem?
  - doesn’t say anything about which points are allowed as endpoints
  - doesn’t give a clear meaning of “draw”
  - doesn’t say what constitutes a “line” in raster world
  - doesn’t say how to measure success of proposed algorithms

Problem Statement

- Given two points *P* and *Q* in XY plane, both with integer coordinates, determine which pixels on raster screen should be on in order to make picture of a unit-width line segment starting at *P* and ending at *Q*

Finding next pixel:

Special case:

- Horizontal Line:

  Draw pixel *P* and increment *x* coordinate value by 1 to get next pixel.

- Vertical Line:

  Draw pixel *P* and increment *y* coordinate value by 1 to get next pixel.

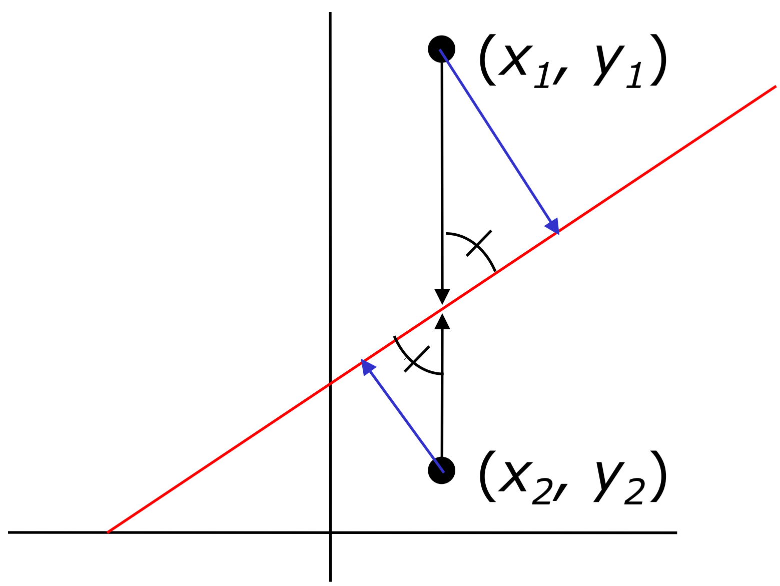
- Diagonal Line:

  Draw pixel *P* and increment both *x* and *y* coordinate by 1 to get next pixel.

- What should we do in general case?
  - Increment x coordinate by 1 and choose point closest to line.
  - But how do we measure “closest”?

Vertical Distance

- Why can we use vertical distance as measure of which point is closer?
  - because vertical distance is proportional to actual distance
  - how do we show this?
  - with similar triangles



- By similar triangles we can see that true distances to line (in blue) are directly proportional to vertical distances to line (in black) for each point
- Therefore, point with smaller vertical distance to line is closest to line

Strategy 1 - Incremental Algorithm (1/2)

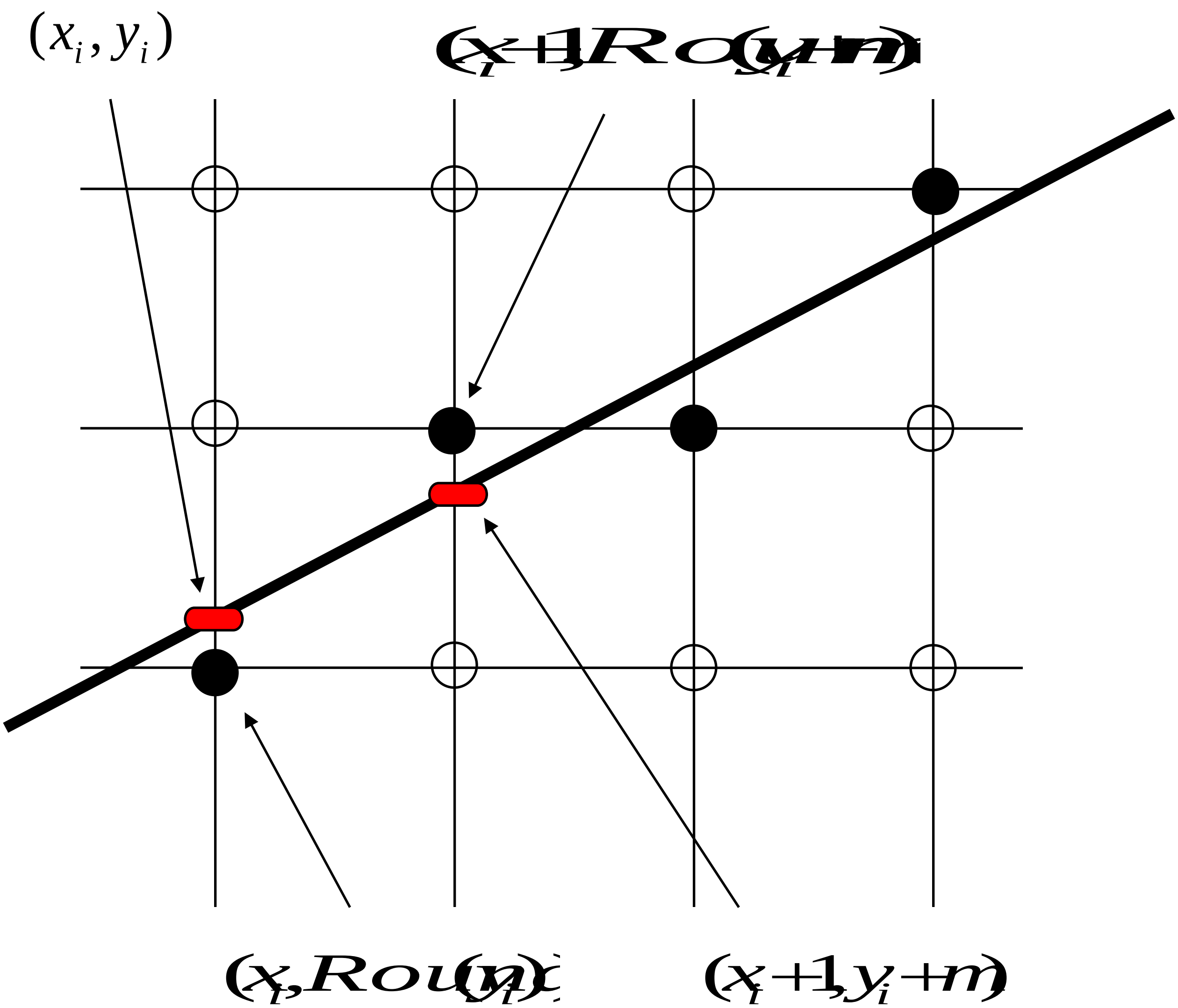
Basic Algorithm

- Find equation of line that connects two points P and Q
- Starting with leftmost point P, increment x_i by 1 to calculate $y_i = m*x_i + B$
  where m = slope, B = y intercept
- Draw pixel at (x_i, Round(y_i))

Incremental Algorithm:

- Each iteration requires a floating-point multiplication
  - Modify algorithm to use deltas
  - $(y_{i+1} - y_i) = m*(x_{i+1} - x_i) + B - B$
  - $y_{i+1} = y_i + m*(x_{i+1} - x_i)$
  - If $\Delta x = 1$, then $y_{i+1} = y_i + m$
- At each step, we make incremental calculations based on preceding step to find next y value

Strategy 1 - Incremental Algorithm (2/2)



Example Code

```
// Incremental Line Algorithm
// Assume x0 < x1

void Line(int x0, int y0,
          int x1, int y1) {
   int    x, y;
   float  dy = y1 - y0;
   float  dx = x1 - x0;
   float  m = dy / dx;

   y = y0;
   for (x = x0; x < x1; x++) {
          WritePixel(x, Round(y));
          y = y + m;
   }
}
```

Problem with Incremental Algorithm:

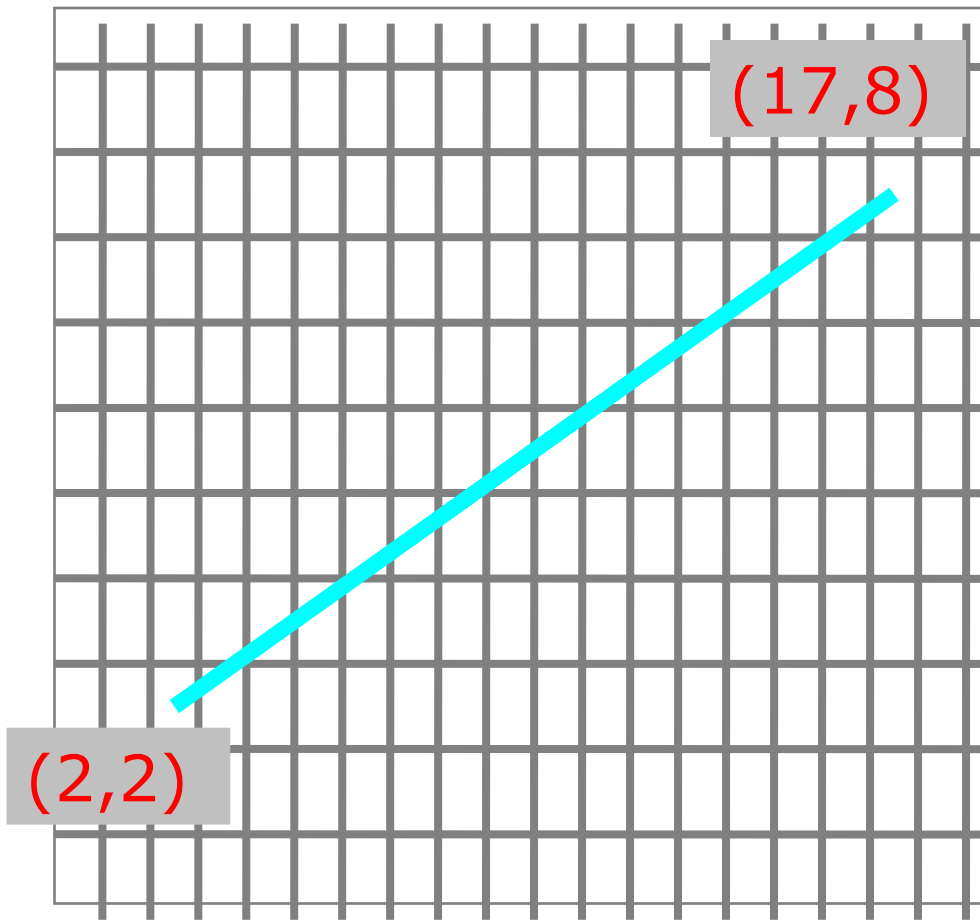
```
void Line(int x0, int y0,
            int x1, int y1) {
   int    x, y;
   float  dy = y1 - y0;
   float  dx = x1 - x0;
   float  m = dy / dx;

   y = y0;
   for (x = x0; x < x1; x++) {
          WritePixel(x, Round(y));
          y = y + m;
   }
}
```

Rounding takes time

Since slope is fractional, need special case for vertical lines

Calculate the coordinates using

Incremental Algorithm?



Discretization - converting a continuous signal into discrete elements.

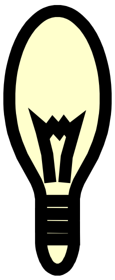
Scan Conversion - converting vertex/edges information into pixel data for display

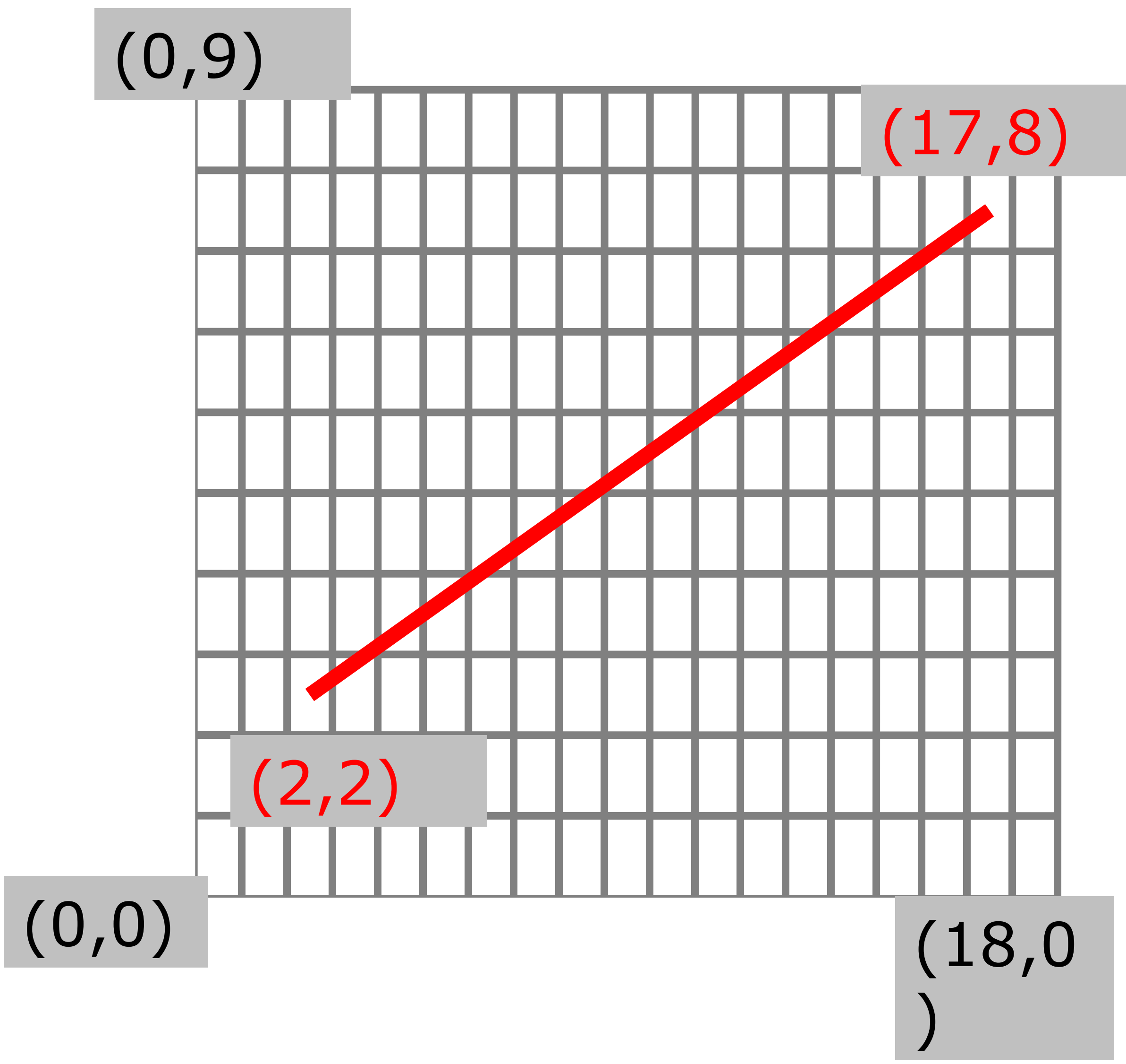
Example 1:

Point1 V:(2,2) C:(255,102,0)

Point2 V:(17,8) C:(255,102,0)

What if colors were different?

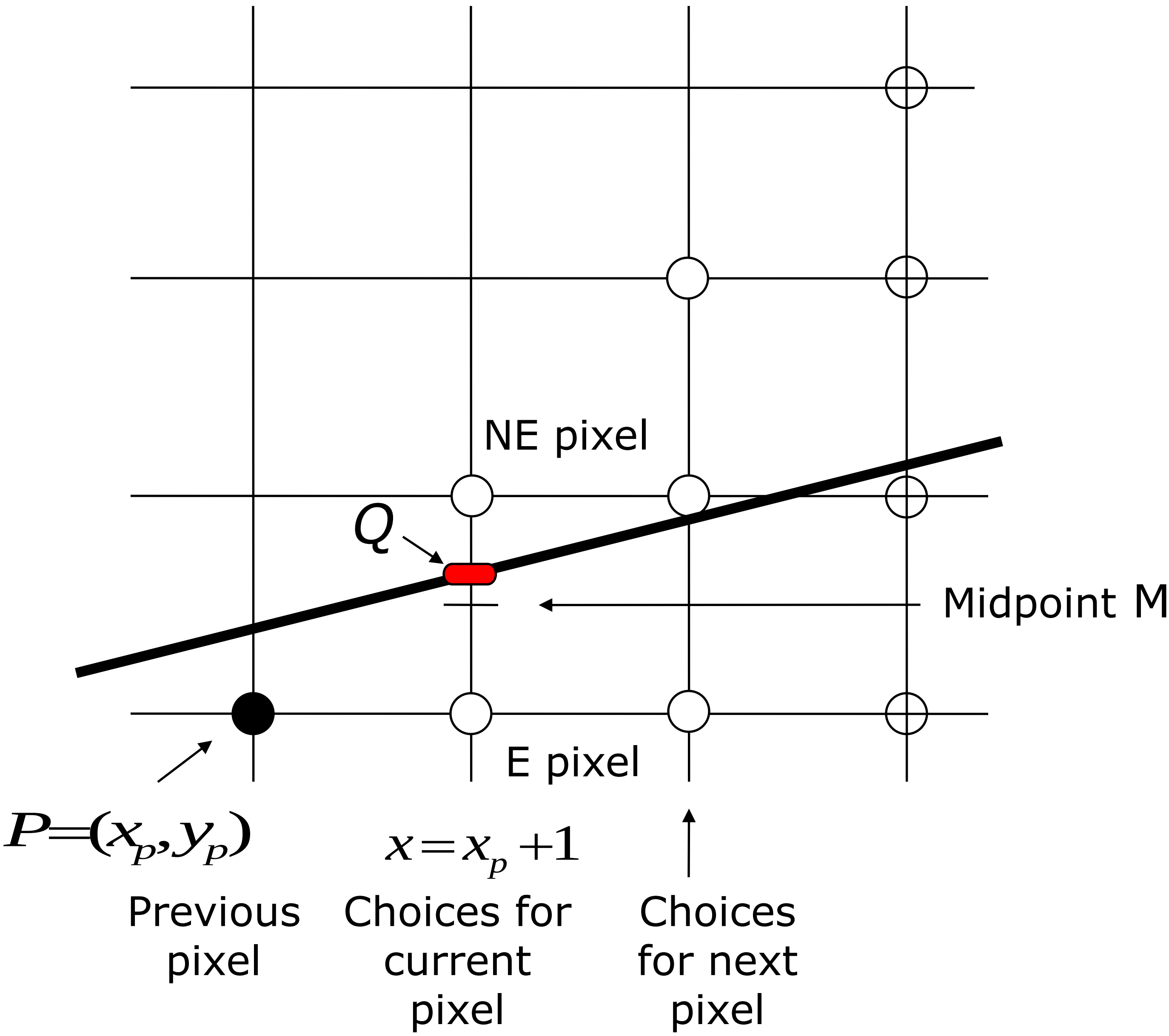




Strategy 2 – Midpoint Line Algorithm (1/3)

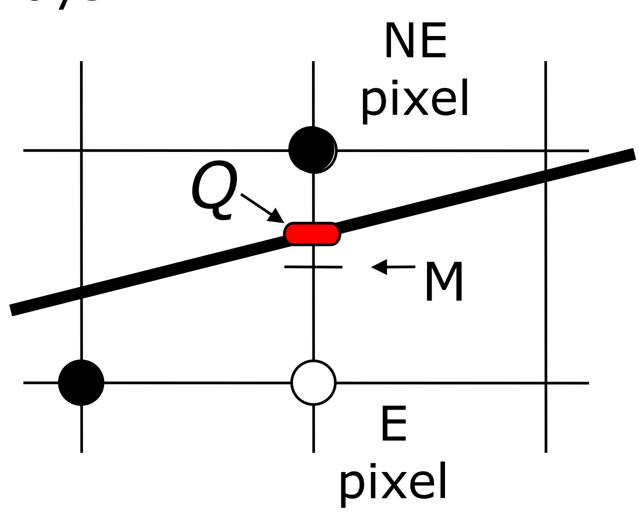
- Assume that line's slope is shallow and positive ($0 < \text{slope} < 1$); other slopes can be handled by suitable reflections about principle axes
- Call lower left endpoint (x_0, y_0) and upper right endpoint (x_1, y_1)
- Assume that we have just selected pixel P at (x_p, y_p)
- Next, we must choose between pixel to right (E pixel), or one right and one up (NE pixel)
- Let Q be intersection point of line being scan-converted and vertical line $x=x_p+1$

Strategy 2 – Midpoint Line Algorithm (2/3)



Strategy 2 – Midpoint Line Algorithm (3/3)

- Line passes between E and NE
- Point that is closer to intersection point *Q* must be chosen
- Observe on which side of line midpoint *M* lies:
  - E is closer to line if midpoint *M* lies above line, i.e., line crosses bottom half
  - NE is closer to line if midpoint *M* lies below line, i.e., line crosses top half
- Error (vertical distance between chosen pixel and actual line) is always <= ½
- Algorithm chooses NE as next pixel for line shown
- Now, need to find a way to calculate on which side of line midpoint lies



Line

Line equation as function *f(x)*:

$$y = mx + B$$

$$y = \frac{dy}{dx}x + B$$

Line equation as implicit function:

$$f(x, y) = ax + by + c = 0$$

for coefficients *a*, *b*, *c*, where *a*, *b* ≠ 0

from above,

$$y\,dx = dy\,x + B\,dx$$

$$dy\,x - y\,dx + B\,dx = 0$$

$$\therefore a = dy,\; b = -dx,\; c = B\,dx$$

Properties (proof by case analysis):

- $f(x_m, y_m) = 0$ when any point *M* is on line
- $f(x_m, y_m) < 0$ when any point *M* is above line
- $f(x_m, y_m) > 0$ when any point *M* is below line
- Our decision will be based on value of function at midpoint *M* at $(x_p + 1, y_p + ½)$

Decision Variable

Decision Variable d:

- We only need sign of $f(x_p + 1, y_p + ½)$ to see where line lies, and then pick nearest pixel
- $d = f(x_p + 1, y_p + ½)$

  - if $d > 0$ choose pixel NE

  - if $d < 0$ choose pixel E

  - if $d = 0$ choose either one consistently

How do we incrementally update d?

- On basis of picking E or NE, figure out location of M for that pixel, and corresponding value d for next grid line
- We can derive d for the next pixel based on our current decision

If E was chosen:

Increment M by one in x direction

$$d_{new} = f(x_p + 2, y_p + ½)$$
$$= a(x_p + 2) + b(y_p + ½) + c$$

$$d_{old} = a(x_p + 1) + b(y_p + ½) + c$$

- $d_{new} - d_{old}$ is the incremental difference ΔE

  $d_{new} = d_{old} + a$
  $\Delta E = a = dy$ (2 slides back)

- We can compute value of decision variable at next step incrementally without computing *F(M)* directly

  $d_{new} = d_{old} + \Delta E = d_{old} + dy$

∀ ΔE can be thought of as correction or update factor to take d_{old} to d_{new}

- It is referred to as <u>forward difference</u>

If NE was chosen:

Increment M by one in both *x* and *y* directions

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

$\forall$ $\Delta NE = d_{new} - d_{old}$

$$d_{new} = d_{old} + a + b$$
$$\Delta NE = a + b = dy - dx$$

- Thus, incrementally,

$$d_{new} = d_{old} + \Delta NE = d_{old} + dy - dx$$

Summary (1/2)

- At each step, algorithm chooses between 2 pixels based on sign of decision variable calculated in previous iteration.

- It then updates decision variable by adding either ΔE or ΔNE to old value depending on choice of pixel. Simple additions only!

- First pixel is first endpoint (x_0, y_0), so we can directly calculate initial value of *d* for choosing between E and NE.

Summary (2/2)

- First midpoint for first $d = d_{start}$ is at $(x_0 + 1, y_0 + ½)$

- $f(x_0 + 1, y_0 + ½)$
  $= a(x_0 + 1) + b(y_0 + ½) + c$
  $= a * x_0 + b * y_0 + c + a + b/2$
  $= \mathrm{f}(x_0, y_0) + a + b/2$

- But (x_0, y_0) is point on line and $f(x_0, y_0) = 0$

- Therefore, $d_{start} = a + b/2 = dy - dx/2$
  - use d_{start} to choose second pixel, etc.

- To eliminate fraction in d_{start} :
  - redefine f by multiplying it by 2; $f(x,y) = 2(ax + by + c)$
  - this multiplies each constant and decision variable by 2, but does not change sign

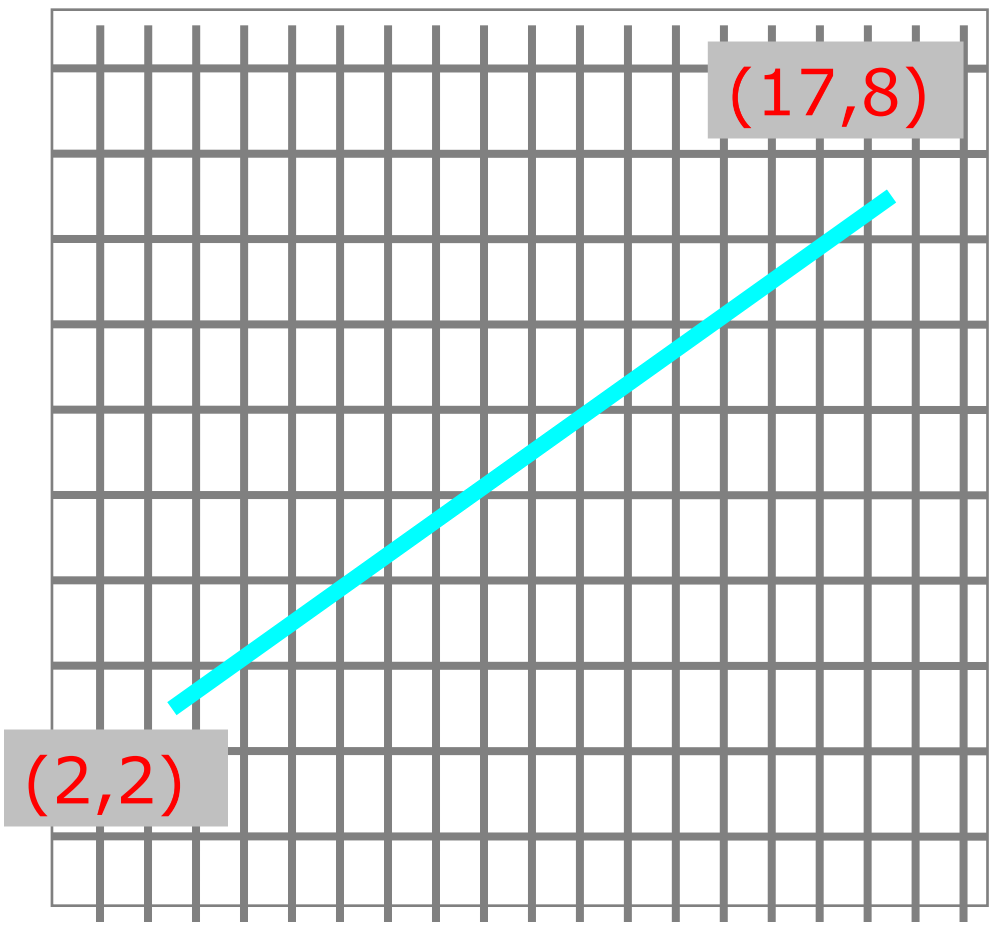
- Bresenham's line algorithm is same but doesn't generalize as nicely to circles and ellipses

Example Code

```
void MidpointLine(int x0, int y0,
                     int x1, int y1) {
   int      dx = x1 - x0;
   int      dy = y1 - y0;
   int      d = 2 * dy - dx;
   int      incrE = 2 * dy;
   int      incrNE = 2 * (dy - dx);
   int      x = x0;
   int      y = y0;

   writePixel(x, y);

   while (x < x1) {
            if (d <= 0) {      // East Case
                     d = d + incrE;
            } else {           // Northeast Case
                     d = d + incrNE;
                     y++;
            }
            x++;
            writePixel(x, y);
   }                 /* while */
}                    /* MidpointLine */
```

Calculate the coordinates using

Midpoint line Algorithm?



Scan Converting Circles

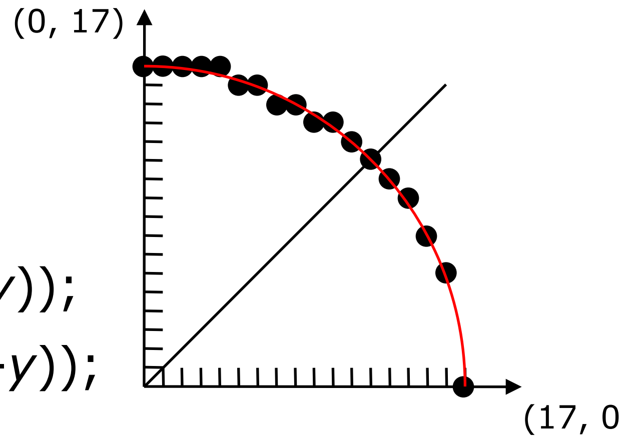
Version 1: <u>really</u> bad

For x = – R to R

    y = sqrt(R * R – x * x);

    Pixel (round(x), round(y));

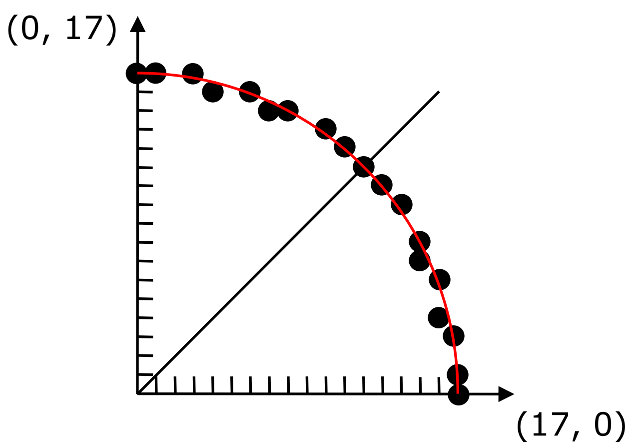
    Pixel (round(x), round(-y));



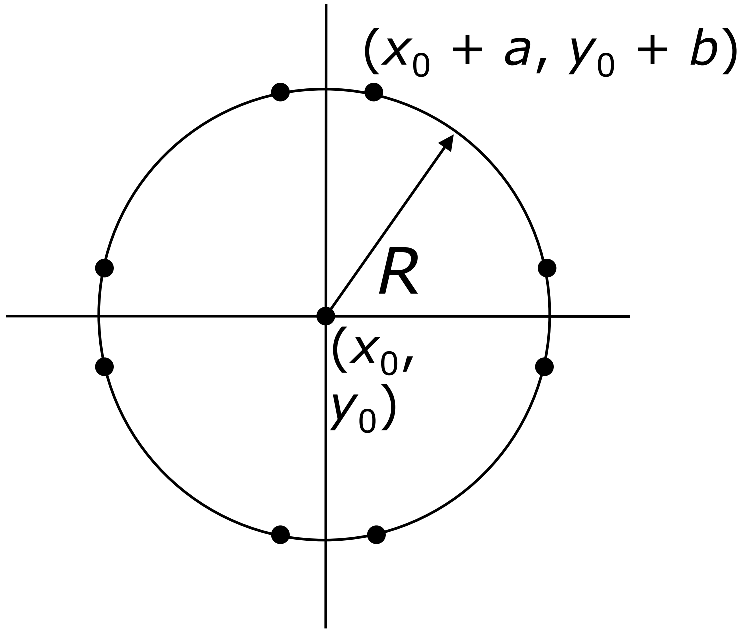
Version 2: slightly less bad

For x = 0 to 360

    Pixel (round (R • cos(x)), round(R • sin(x)));



Version 3 — Use Symmetry



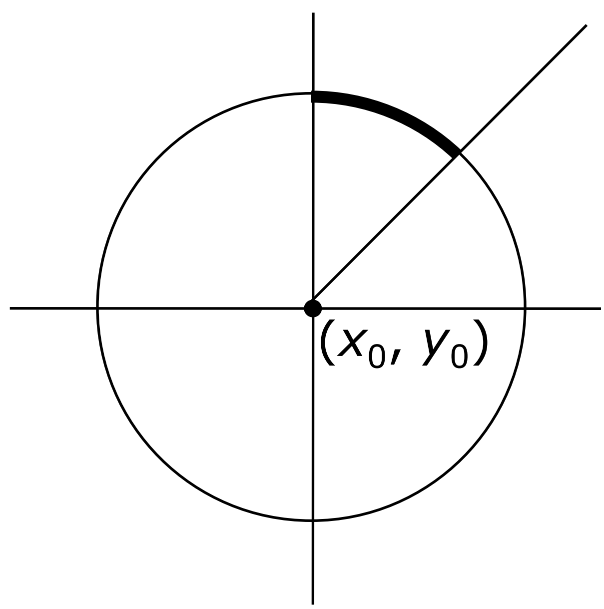
$(x\text{-}x_0)^2 + (y\text{-}y_0)^2 = R^2$

- Symmetry: If $(x_0 + a, y_0 + b)$ is on circle
  - also $(x_0 \pm a, y_0 \pm b)$ and $(x_0 \pm b, y_0 \pm a)$;
    hence 8-way symmetry.

- Reduce the problem to finding the pixels for 1/8 of the circle

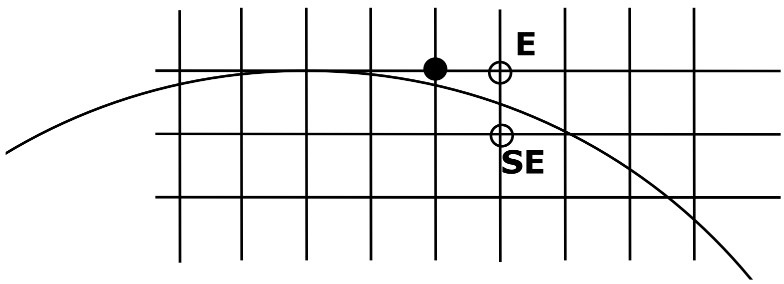
Using the Symmetry

- Scan top right 1/8 of circle of radius *R*



- Circle starts at $(x_0, y_0 + R)$
- Let's use another incremental algorithm with decision variable evaluated at midpoint

Sketch of Incremental Algorithm




```
x = x0; y = y0 + R; Pixel(x, y);
for (x = x0+1; (x - x0) > (y - y0); x++) {
   if (decision_var < 0) {
        /* move east */
        update decision_var;
   }
   else {
        /* move south east */
        update decision_var;
        y--;
   }
   Pixel(x, y);
}
```

- Note: can replace all occurrences of x_0, y_0 with 0, 0 and Pixel ($x_0 + x$, $y_0 + y$) with Pixel (x, y)
- Shift coordinates by ($-x_0$, $-y_o$)

What we need for Incremental Algorithm

- Decision variable
  - negative if we move E, positive if we move SE (or vice versa).

- Follow line strategy: Use implicit equation of circle

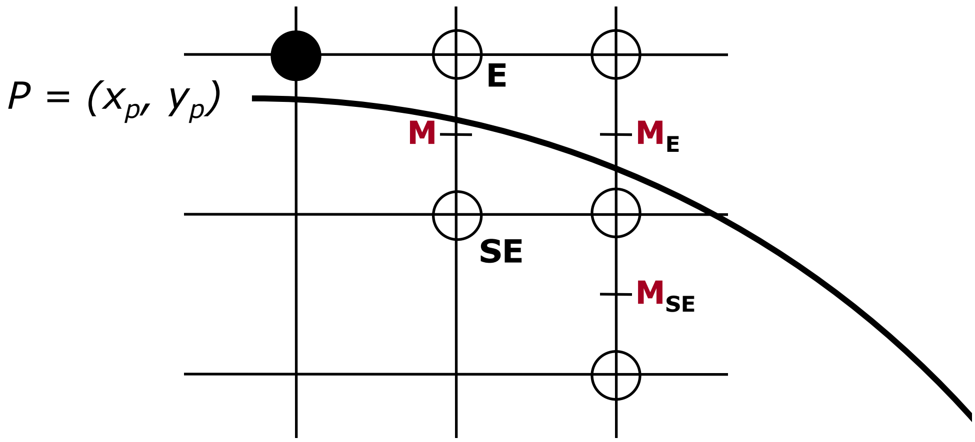
  $$f(x,y) = x^2 + y^2 - R^2 = 0$$

  $f(x,y)$ is zero on circle, negative inside, positive outside

- If we are at pixel (x, y)
  - examine $(x + 1, y)$ and $(x + 1, y - 1)$

- Compute f at the midpoint

Decision Variable



- Evaluate $f(x,y) = x^2 + y^2 - R^2$ at the point

$$\left(x+1, y-\frac{1}{2}\right)$$

- We are asking: "Is

$$f\left(x+1, y-\frac{1}{2}\right) = (x+1)^2 + \left(y-\frac{1}{2}\right)^2 - R^2$$

positive or negative?" (it is zero on circle)

- If **negative**, midpoint inside circle, **choose E**
  - *vertical* distance to the circle is less at

(x + 1, y) than at (x + 1, y–1).

- If **positive**, opposite is true, **choose SE**

The right decision variable?

- Decision based on vertical distance
- Ok for lines, since *d* and d_{vert} are proportional
- For circles, not true:

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

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

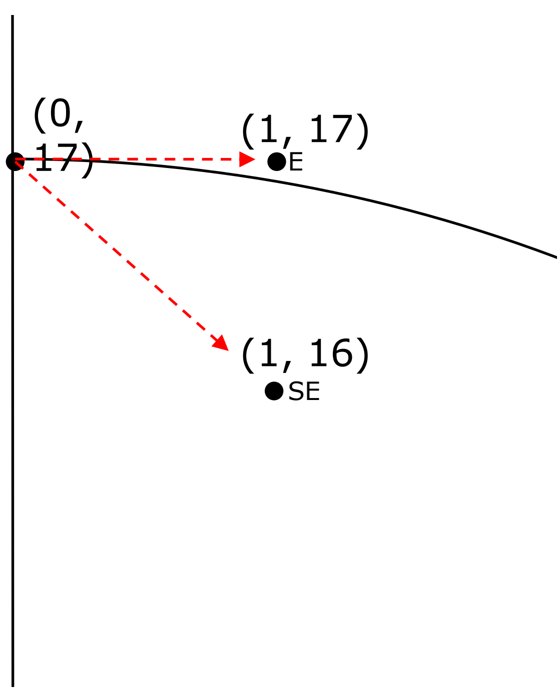
- Which *d* is closer to zero? (i.e. which of the two values below is closer to *R*):

$$\sqrt{(x+1)^2+y^2} \quad or \quad \sqrt{(x+1)^2+(y-1)^2}$$

- We could ask instead:
  "Is $(x + 1)^2 + y^2$ or $(x + 1)^2 + (y - 1)^2$ closer to R^2?"

- The two values in equation above differ by

$$[(x+1)^2 + y^2] - [(x+1)^2 + (y-1)^2] = 2y - 1$$



$f_E = 1^2 + 17^2 = 290$

$f_{SE} = 1^2 + 16^2 = 257$

$f_E - f_{SE} = 290 - 257 = 33$

$2y - 1 = 2(17) - 1 = 33$

Incremental Computation, Again (1/2)

- How to compute the value of

$$f(x,y) = (x+1)^2 + \left(y - \frac{1}{2}\right)^2 - R^2$$

at successive points?

- Answer: Note that

is just $$f(x+1,y) - f(x,y)$$

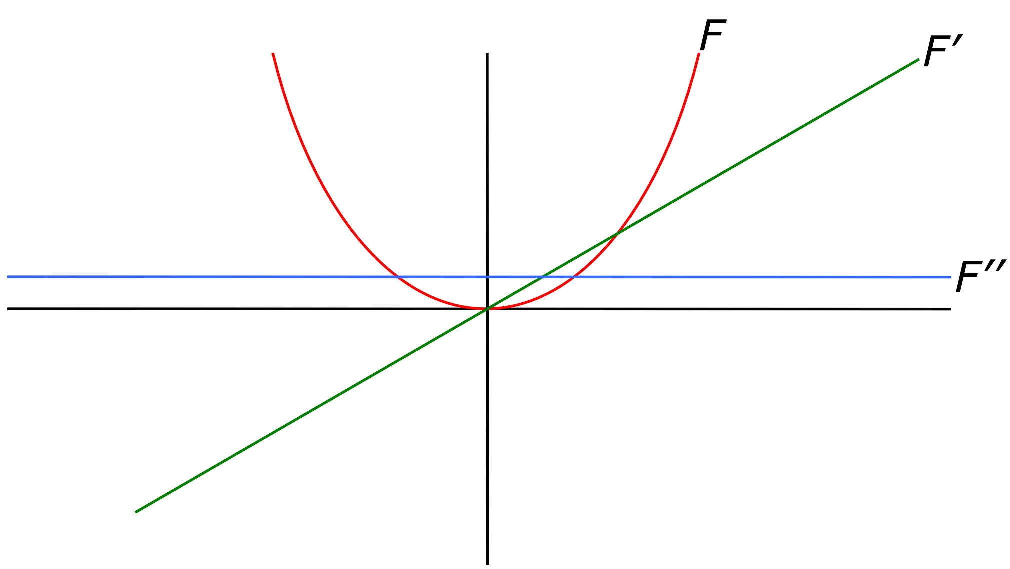
and that $$\Delta_E(x,y) = 2x + 3$$

is just $$f(x+1,y-1) - f(x,y)$$

$$\Delta_{SE}(x,y) = 2x + 3 - 2y + 2$$

Incremental Computation (2/2)

- If we move E, update by adding $2x + 3$
- If we move SE, update by adding
  $2x + 3 - 2y + 2$.
- Forward differences of a 1st degree polynomial are constants and those of a 2nd degree polynomial are 1st degree polynomials
  - this "first order forward difference," like a partial derivative, is one degree lower



Second Differences (1/2)

- The function $\Delta_E(x,y) = 2x+3$ is linear, hence amenable to incremental computation:

$$\Delta_E(x+1,y) - \Delta_E(x,y) = 2$$

$$\Delta_E(x+1,y-1) - \Delta_E(x,y) = 2$$

- Similarly

$$\Delta_{SE}(x+1,y) - \Delta_{SE}(x,y) = 2$$

$$\Delta_{SE}(x+1,y-1) - \Delta_{SE}(x,y) = 4$$

Second Differences (2/2)

- For any step, can compute new $\Delta_E(x, y)$ from old $\Delta_E(x, y)$ by adding appropriate second constant increment – update delta terms as we move.
  - This is also true of $\Delta_{SE}(x, y)$

- Having drawn pixel (a,b), decide location of new pixel at $(a + 1, b)$ or $(a + 1, b - 1)$, using previously computed $\Delta(a, b)$.

- Having drawn new pixel, must update $\Delta(a, b)$ for next iteration; need to find either $\Delta(a + 1, b)$ or $\Delta(a + 1, b - 1)$ depending on pixel choice

- Must add $\Delta_E(a, b)$ or $\Delta_{SE}(a, b)$ to $\Delta(a, b)$

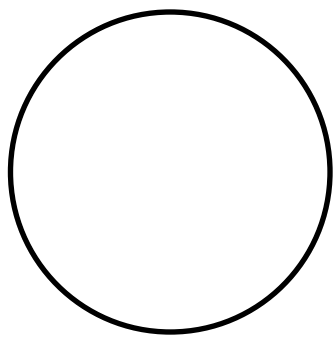
- So we…
  - Look at $\Delta(i)$ to decide which to draw next, update x and y
  - Update d using $\Delta_E(a,b)$ or $\Delta_{SE}(a,b)$
  - Update each of $\Delta_E(a,b)$ and $\Delta_{SE}(a,b)$ for future use
  - Draw pixel

Midpoint Eighth Circle Algorithm

```
MEC (R) /* 1/8th of a circle w/ radius R */
{
int x = 0, y = R;
int delta_E, delta_SE;
float decision;
delta_E = 2*x + 3;
delta_SE = 2(x-y) + 5;
decision = (x+1)*(x+1) + (y + 0.5)*(y + 0.5) -R*R;
Pixel(x, y);
while( y > x ) {
        if (decision > 0) {/* Move east */
                decision += delta_E;
                delta_E += 2; delta_SE += 2; /*Update
delta*/
        }
        else {/* Move SE */
                y--;
                decision += delta_SE;
                delta_E += 2; delta_SE += 4; /*Update
delta*/
        }
        x++;
        Pixel(x, y);
}
}
```

Analysis

- Uses floats!
- 1 test, 3 or 4 additions per pixel
- Initialization can be improved
- Multiply everything by 4 $\rightarrow$ No Floats!
  - Makes the components even, but sign of decision variable remains same

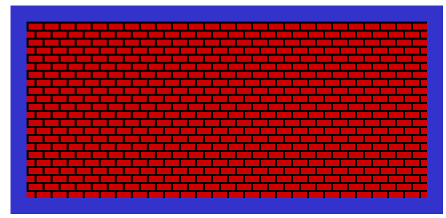


Questions

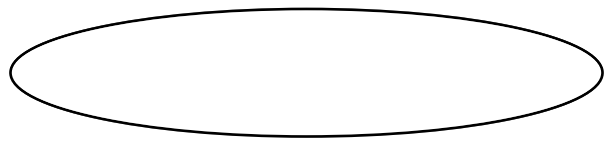
- Are we getting <u>all</u> pixels whose distance from the circle is less than ½?
- Why is ***y > x*** the right stopping criterion?
- What if it were an ellipse?

Other Scan Conversion Problems

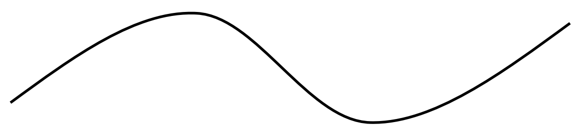
- Patterned primitives



- Aligned Ellipses



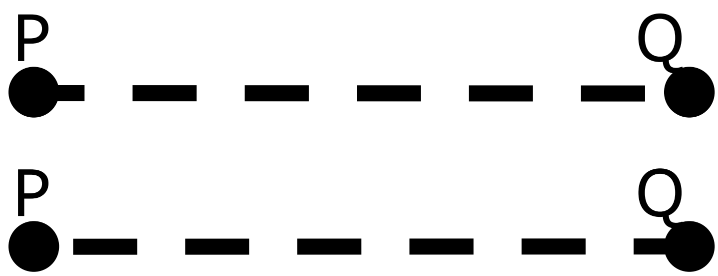
- Non-integer primitives



- General conics

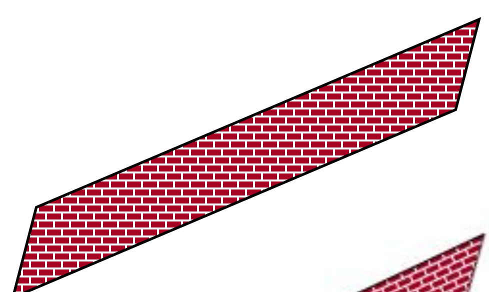
Patterned Lines

- Patterned line from *P* to *Q* is <u>not</u> same as patterned line from *Q* to *P*.

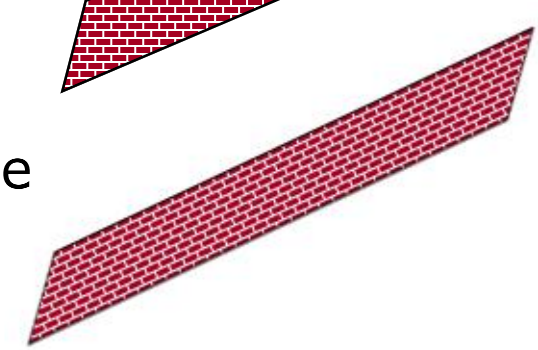


- Patterns can be geometric or cosmetic
  - Cosmetic: Texture applied after transformations
  - Geometric: Pattern subject to transformations

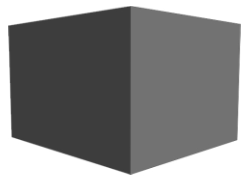
Cosmetic patterned line



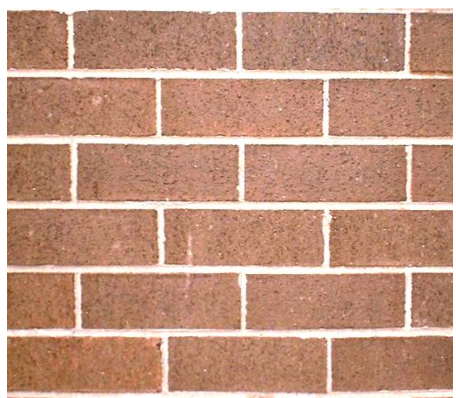
Geometric patterned line



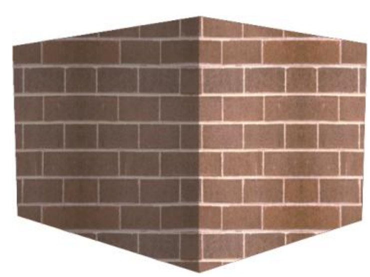
Geometric Pattern vs. Cosmetic Pattern



+



Geometric
(Perspectivized/Filtered)



Cosmetic
(Contact Paper)

Aligned Ellipses

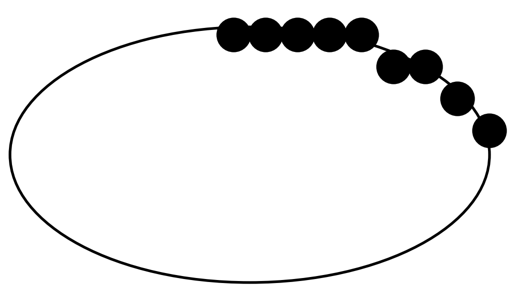
- Equation is

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

i.e,

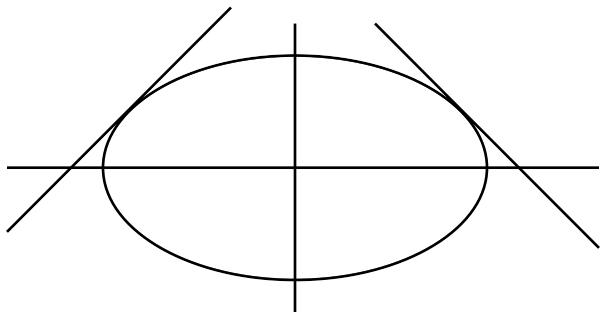
$$b^2x^2 + a^2y^2 = a^2b^2$$

- Computation of Δ_E and Δ_{SE} is similar
- Only 4-fold symmetry
- When do we stop stepping horizontally and switch to vertical?



Direction Changing Criterion (1/2)

- When absolute value of slope of ellipse is more than 1:



- How do you check this? At a point (x,y) for which $f(x,y) = 0$, a vector perpendicular to the level set is $\nabla f(x,y)$ which is

$$\left[\frac{\partial f}{\partial x}(x,y), \frac{\partial f}{\partial y}(x,y)\right]$$

- This vector points more right than up when

$$\frac{\partial f}{\partial x}(x,y) - \frac{\partial f}{\partial y}(x,y) > 0$$

Direction Changing Criterion (2/2)

Major axis lies on x axis

- In our case,

$$\frac{\partial f}{\partial x}(x,y) = 2b^2x$$

and

$$\frac{\partial f}{\partial y}(x,y) = 2a^2y$$

so we check for

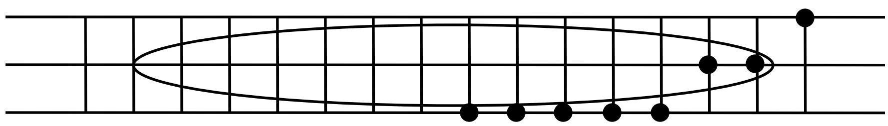
$$2b^2x - 2a^2y > 0$$

i.e.

$$a^2x - b^2y > 0$$

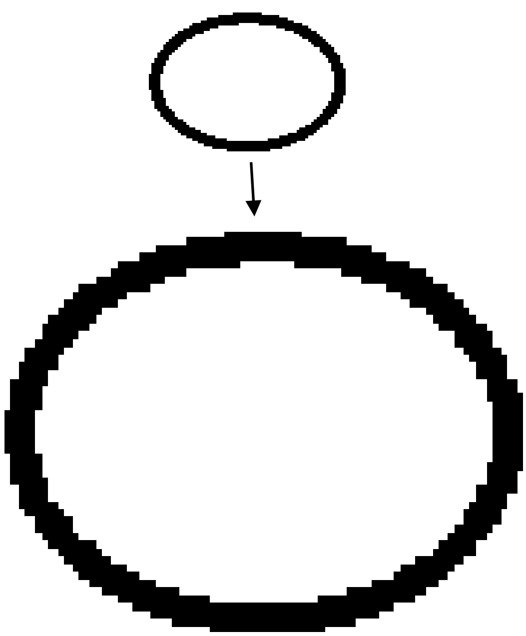
- This, too, can be computed incrementally

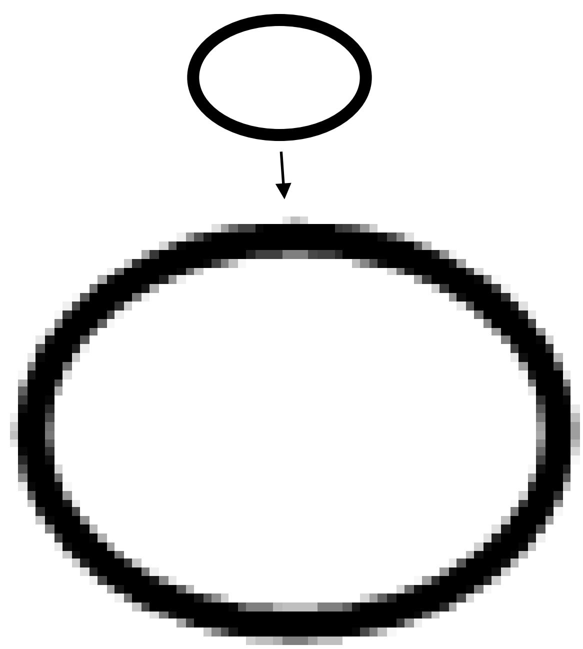
Problems with Aligned Ellipses



- Now in ENE octant, <u>not</u> ESE octant

- This problem is artifact of *aliasing* – <u>much</u> more on this later





Generic Polygons