Review of Last Time (1/2)

- Wanted to scan convert the line from $P = (x_0, y_0)$ to $Q = (x_1, y_1)$
- Decided to restrict our attention, by symmetry, to cases where $x_0 \le x_1$, and $0 \le slope \le 1$.
- Decided, on no compelling basis, to seek exactly one pixel in column.
- Decided to choose the pixel closest to the centerline of the unit-width segment
- Expressed the segment in the form $Ax + By + C = 0$ with $B > 0$.
- Found that the value of $Aa + Bb + C$ was a *proxy* for the signed distance from (a, b) to the line
- Looked at $\left(a+1, b+\frac{1}{2}\right)$ to decide whether to move "east" or "northeast" for next pixel. Set

$$D = A(a+1) + B\left(b + \frac{1}{2}\right) + C$$

Review (2/2)

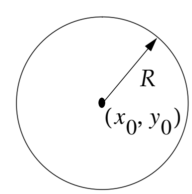
- Found that $D(a+1, b)$ could be computed from $D(a, b)$ by one addition. Same for $D(a+1, b+1)$
- Devised "midpoint" algorithm.
- Made it efficient by doubling everything, getting rid of floats

The Improved Code

```
midpoint_special_case(int x0, int y0,
         int x1, int y1)
{
   int   x, y, A, B, C, D, delta_E,
       delta_NE;
   A = y0 - y1; B = x1 - x0;
   C = x0 * y1 - x1 * y0;
   D = 2*A*x0 + 2*B*y0 + 2*C + B;
   Pixel(x0, y0);
   x = x0 + 1; y = y0;
   delta_E = 2*A;
   delta_NE = delta_E + 2*B;

   for(; x <= x1; x++) {
      if (D <0) { /*Draw top pixel*/
         y++;
         D += delta_NE;
      }
      else { /* D>0 draw lower pixel */
         D += delta_E;
      }
      Pixel(x, y);
   }
}
```


Scan Converting Circles

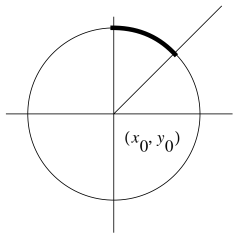


$$(x - x_0)^2 + (y - y_0)^2 = R^2$$

- Symmetry: If $(x_0 + a, y_0 + b)$ is on the circle, so are $(x_0 \pm a, y_0 \pm b)$ and $(x_0 \pm b, y_0 \pm a)$, hence there's an 8-way symmetry.
- But in a practical setting of considering pixel values, it depends on the fact that x_0 and y_0 are integers.

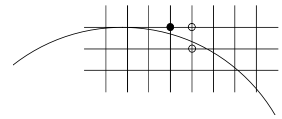
Using the Symmetry

- We will scan top right 1/8 of circle of radius R



- It starts at $(x_0, y_0 + R)$

Sketch of Algorithm



```
y = y0 + R; x = x0; 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 occurences of x_0 and y_0 with 0,0 and Pixel(x, y) with Pixel(x_0+x, y_0+y).
- Essentially a *shift of coordinates*

Bad Alternatives

- Version 1:

```
For x = 0 to 360
Pixel(round(R*cos(x)),round(R*sin(x)));
```

- Version 2

```
For x = -R to R
   y = sqrt(R*R - x*x);
   Pixel(round(x), round(y));
   Pixel(round(x), round(-y));
```

- HW: explain why

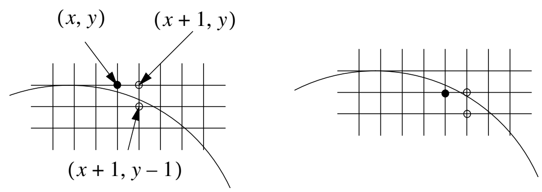
What We Need

- Need a decision variable, i.e., something that is negative if we should move E, positive if we should move SE (or vice versa).
- Follow previous lead: the implicit equation of circle

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

  is zero on the circle, negative inside it, positive outside.
- If we are at pixel (x, y), examine $(x + 1, y)$ and $(x + 1, y - 1)$.
- Again use the trick of average of h at these points = h(midpoint).

Decision Variable



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

  at the point $\left(x+1, y-\frac{1}{2}\right)$
- What we <u>are</u> asking is this: "Is

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

  positive or negative?"
- If it is negative there, this mid point is inside the circle, so the *vertical* distance to the circle is less at $(x+1, y)$ than at $(x+1, y-1)$.
- If it is positive, the opposite is true.

Is this the right decision variable?

- It makes our decision based on vertical distance.
- For lines, that was ok, since d and d_{vert} were proportional.
- For circles, no longer true:

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

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

- We ask which d is closer to zero, i.e., which of the two values below is closer to R:

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

Alternate Phrasing (1/3)

- 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

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

Alternate Phrasing (2/3)

- So the second value, which is always the lesser

  is *closer* if its difference from R^2 is less

  than $\left(\frac{1}{2}\right)(2y-1)$ i.e., if

$$R^2 - [(x+1)^2 + (y-1)^2] < \frac{1}{2}(2y-1)$$

then

$$0 < y - \frac{1}{2} + (x+1)^2 + (y-1)^2 - R^2$$

so

$$0 < (x+1)^2 + y^2 - 2y + 1 + y - \frac{1}{2} - R^2$$

so

$$0 < (x+1)^2 + y^2 - y + \frac{1}{2} - R^2$$

so

$$0 < (x+1)^2 + \left(y - \frac{1}{2}\right)^2 + \frac{1}{4} - R^2$$

Alternate Phrasing (3/3)

- So the *radial* distance decision is whether

$$d1 = (x+1)^2 + \left(y - \frac{1}{2}\right)^2 + \frac{1}{4} - R^2$$

is positive or negative,

- And the *vertical* distance decision is whether

$$d2 = (x+1)^2 + \left(y - \frac{1}{2}\right)^2 - R^2$$

is positive or negative; *d1* and *d2* are $\frac{1}{4}$ apart.

- *d2* is an integer, and is positive only if $d2 + \frac{1}{4}$ is positive (except special case where $d2 = 0$). Hence, aside from ambiguous cases, the two are the same.

Incremental Computation, Again (1/2)

- How should we compute the value of

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

at successive points?

- Answer: Note that

$$h(x+1, y) - h(x, y)$$

is just

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

and that

$$h(x+1, y-1) - h(x, y)$$

is just

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

Incremental Computation (2/2)

- So if we move east, update by adding $2x + 3$
- And if we move SE, update by adding $2x + 3 - 2y + 2$.
- Note that the forward differences of a 1st degree polynomial were 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. Let’s make use of this property.

Second Differences (1/2)

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

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

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

- 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)

- So for any step, we can compute new $\Delta_E(x, y)$ from old $\Delta_E(x, y)$ by adding an appropriate second constant increment -- we update the update term as we move.
- Having previously drawn pixel (a, b), in current iteration we decide between drawing pixel at $(a + 1, b)$ and $(a + 1, b - 1)$, using previously computed $d(a, b)$.
- Having drawn the pixel, we must update $d(a, b)$ for use next time; we'll need to find either $d(a + 1, b)$ or $d(a + 1, b - 1)$ depending on which pixel we chose.
- Will require adding $\Delta_E(a,b)$ or $\Delta_{SE}(a,b)$ to $d(a, b)$.
- So we ...
- Look at $d(a, b)$ 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);
   for(; x < y; x++){
      if (decision > 0) { /* Move east */
         decision += delta_E;
         delta_E += 2; delta_SE += 2;
      }
      else { /* move SE */
         y--;
         decision += delta_SE;
         delta_E += 2; delta_SE += 4;
      }
      Pixel(x, y);
   }
}
```


Analysis

- Uses a float!
- 1 test, 3 or 4 additions per pixel.
- Initialization can be improved
- Multiply everything by 4 ===> No Floats!

Questions

- Are we getting <u>all</u> pixels whose distance from the circle is less than 1/2?
- Why is "x < y" 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 can be from a background or a pattern sequence.

Aligned Ellipses

- Equation is

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

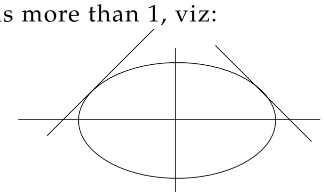
i.e.,

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

- 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 the absolute value of the slope of the ellipse is more than 1, viz:



- 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

$$[\frac{\partial F}{\partial x}(x, y), \frac{\partial F}{\partial y}(x, y)]$$

- This vector points more <u>right</u> than <u>up</u> when

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

Direction Changing Criterion (2/2)

- In our case,

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

and

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

so we check for

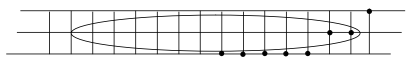
$$2a^2x - 2b^2y > 0$$

i.e.

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

- This, too, can be computed incrementally.

Problems with Aligned Ellipses



- Now in ENE octant, not ESE octant. This problem is due to aliasing - much more on this later.

Non-Integer Primitives

- Initialization is harder
- Endpoints are hard, too.
  - making Line (P,Q) and Line (Q,R) join properly is a good test.
- Symmetry is lost
  - HW: find a non-integer circle which, when scanned by "by-eye" midpoint algorithm, exhibits no symmetry.

General Conics

- Very hard--the octant-changing test is tougher, the difference computations are tougher, etc. Do it only if you have to.
- Note that when we get to drawing grey-scale conics, we will find that this task is easier than drawing B/W conics. If we had solved this problem first, life would have been easier.