



CIS 636

Introduction to Computer Graphics

CG Basics 2 of 8: Rasterization and 2-D Clipping

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course pages: <http://snipurl.com/1y5gc>

Course web site: <http://www.kddresearch.org/Courses/CIS636>

Instructor home page: <http://www.cis.ksu.edu/~bhsu>

Readings:

Sections 2.4 – 2.6, Eberly 2^o – see <http://snurl.com/1ye72>

Chapter 3, Foley, J. D., VanDam, A., Feiner, S. K., & Hughes, J. F. (1991).

Computer Graphics, Principles and Practice, Second Edition in C.

NeHe tutorials: <http://nehe.gamedev.net> (#2: polygons, 21: lines, 26: clipping)



Lecture Outline

- **Scan Conversion of Lines**
 - * Naïve algorithm
 - * Midpoint algorithm (*aka* Bresenham's)
 - * Method of forward differences
- **Scan Conversion of Polygons**
 - * Scan line interpolation
 - * Why it's important: basis of shading/texturing, z-buffering
- **Scan Conversion of Circles and Ellipses**
- **Aliasing and Antialiasing**
 - * Basic problem defined
 - * Approaches





Online Recorded Lectures for CIS 636 *Introduction to Computer Graphics*

- **Project Topics for CIS 636**
- **Computer Graphics Basics (8)**
 - * 1. **Mathematical Foundations – Week 2**
 - * 2. **Rasterizing and 2-D Clipping – Week 3**
 - * 3. **OpenGL Primer 1 of 3 – Week 3**
 - * 4. **Detailed Introduction to 3-D Viewing – Week 4**
 - * 5. **OpenGL Primer 2 of 3 – Week 5**
 - * 6. **Polygon Rendering – Week 6**
 - * 7. **OpenGL Primer 3 of 3 – Week 8**
 - * 8. **Visible Surface Determination – Week 9**
- **Recommended Background Reading for CIS 636**
- **Shared Lectures with CIS 736 (*Computer Graphics*)**
 - * **Regular in-class lectures (35) and labs (7)**
 - * **Guidelines for paper reviews – Week 7**
 - * **Preparing term project presentations, demos for graphics – Week 11**



Scan Converting Lines

Line Drawing

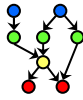
- Draw a line on a raster screen between two points
- What's wrong with the statement of the problem?
 - * it doesn't say anything about which points are allowed as endpoints
 - * it doesn't give a clear meaning to "draw"
 - * it doesn't say what constitutes a "line" in the raster world
 - * it doesn't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

Problem Statement

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

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.





Finding the next pixel

Special case:

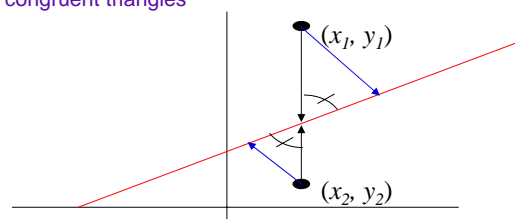
- Horizontal Line:
Draw pixel P and increment the x coordinate value by one to get the next pixel.
- Vertical Line:
Draw pixel P and increment the y coordinate value by one to get the next pixel.
- Perfect Diagonal Line:
Draw pixel P and increment both the x and the y coordinate by one to get the next pixel.
- What should we do in the general case?
 - * Increment the x coordinate by 1 and choose the point closest to the line.
 - * But how do we measure "closest"?

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Vertical Distance

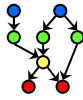
- Why can we use the vertical distance as a measure of which point is closer?
 - * because the vertical distance is proportional to the actual distance
 - * how do we show this?
 - * with congruent triangles



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

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.





Strategy 1: Incremental Algorithm [1]

The Basic Algorithm

- Find the equation of the line that connects the two points P and Q
- Starting with the leftmost point P , increment x_i by 1 to calculate $y_i = mx_i + B$
where $m = \text{slope}$, $B = y \text{ intercept}$
- Intensify the pixel at $(x_i, \text{Round}(y_i))$ where
 $\text{Round}(y_i) = \text{Floor}(0.5 + y_i)$

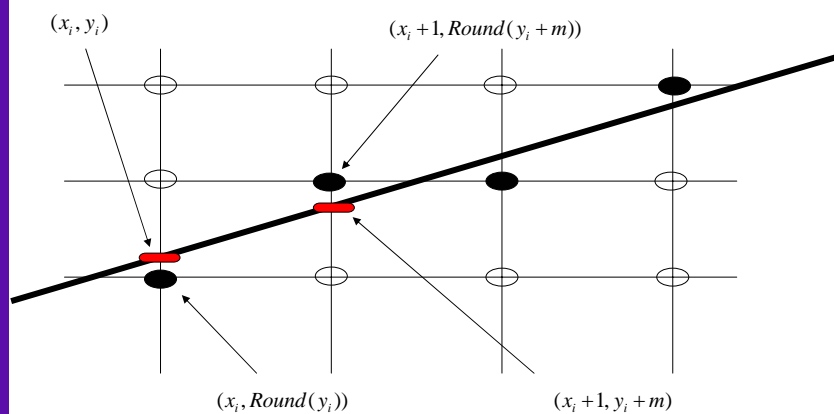
The Incremental Algorithm:

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

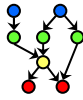
Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Strategy 1: Incremental Algorithm [2]



Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Example Code

```
// Incremental Line Algorithm
// Assumes -1 <= m <= 1, x0 < x1

void Line(int x0, int y0,
         int x1, int y1, int value) {
    int    x;
    float  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), value);
        y = y + m;
    }
}
```

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Problems with Incremental Algorithm

```
void Line(int x0, int y0,
         int x1, int y1, int value)
{
    int    x;
    float  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),
        value);
        y = y + m;
    }
}
```

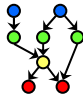
Rounding takes time

Since slope is fractional, need special case for vertical lines

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



ai



Strategy 2: Midpoint Line Algorithm [1]

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

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.

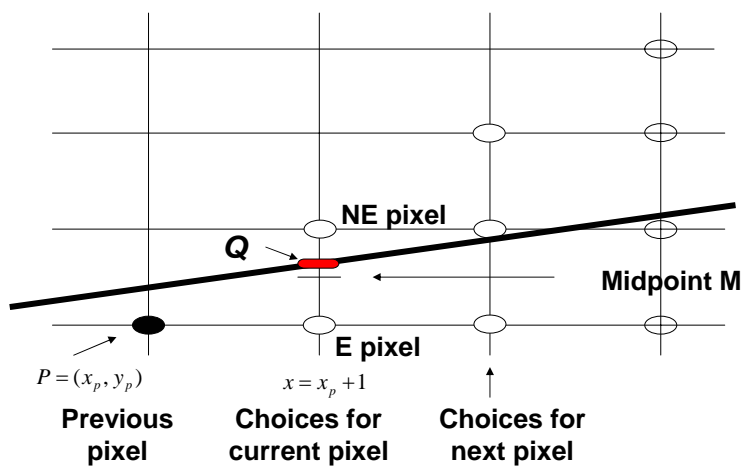
CIS 636/736: (Introduction to) Computer Graphics

CG Basics 2 of 8: Rasterization

Computing & Information Sciences
Kansas State University



Strategy 2: Midpoint Line Algorithm [2]



Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.

CIS 636/736: (Introduction to) Computer Graphics

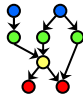
CG Basics 2 of 8: Rasterization

Computing & Information Sciences
Kansas State University



幻灯片 11

a1 last line has a horrible orphan - look for those!
avd, 9/27/2004



Strategy 2: Midpoint Line Algorithm [3]

- The line passes between E and NE
- The point that is closer to the intersection point Q must be chosen
- Observe on which side of the line the midpoint M lies:
 - * E is closer to the line if the midpoint M lies above the line, i.e., the line crosses the bottom half
 - * NE is closer to the line if the midpoint M lies below the line, i.e., the line crosses the top half
- The error, the vertical distance between the chosen pixel and the actual line, is always $\leq \frac{1}{2}$
- The algorithm chooses NE as the next pixel for the line shown
- Now, find a way to calculate on which side of the line the midpoint lies

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



The Line

Line equation as a function $f(x)$:

- $y = f(x) = m \cdot x + B = \frac{dy}{dx} \cdot x + B$

Line equation as an implicit function:

- $F(x, y) = a \cdot x + b \cdot y + c = 0$ for coefficients a, b, c , where $a, b \neq 0$
 from above, $y \cdot dx = \frac{dy}{dx} \cdot x + B \cdot dx$
 so $a = \frac{dy}{dx}$, $b = -dx$, $c = B \cdot dx$,
 $a > 0$ for $y_0 < y_1$

Properties (proof by case analysis):

- $F(x_m, y_m) = 0$ when any point M is on the line
- $F(x_m, y_m) < 0$ when any point M is above the line
- $F(x_m, y_m) > 0$ when any point M is below the line
- Our decision will be based on the value of the function at the midpoint M at $(x_p + 1, y_p + \frac{1}{2})$

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.





Decision Variable

Decision Variable d :

- We only need the sign of $F(x_p + 1, y_p + \frac{1}{2})$ to see where the line lies, and then pick the nearest pixel
- $d = F(x_p + 1, y_p + \frac{1}{2})$
 - 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 the basis of picking E or NE, figure out the location of M for the next grid line, and the corresponding value of $d = F(M)$ for that grid line

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



E Pixel Chosen

M is incremented by one step in the x direction

$$d_{new} = F(x_p + 2, y_p + \frac{1}{2})$$

$$= a(x_p + 2) + b(y_p + \frac{1}{2}) + c$$

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

- Subtract d_{old} from d_{new} to get the incremental difference ΔE

$$d_{new} = d_{old} + a$$

$$\Delta E = a = dy$$

- Gives 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 the correction or update factor to take d_{old} to d_{new}
- It is referred to as the forward difference

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



NE Pixel Chosen

M is incremented by one step each in both the x and y directions

$$\begin{aligned}d_{new} &= F(x_p + 2, y_p + 3/2) \\ &= a(x_p + 2) + b(y_p + 3/2) + c\end{aligned}$$

- Subtract d_{old} from d_{new} to get the incremental difference

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

- Thus, incrementally,

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

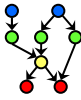
Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Summary [1]

- At each step, the algorithm chooses between 2 pixels based on the sign of the decision variable calculated in the previous iteration.
- It then updates the decision variable by adding either ΔE or ΔNE to the old value depending on the choice of pixel. Simple additions only!
- First pixel is the first endpoint (x_0, y_0) , so we can directly calculate the initial value of d for choosing between E and NE.

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Summary [2]

- First midpoint for first $d = d_{start}$ is at $(x_0 + 1, y_0 + \frac{1}{2})$
- $F(x_0 + 1, y_0 + \frac{1}{2})$
 $= a(x_0 + 1) + b(y_0 + \frac{1}{2}) + c$
 $= a * x_0 + b * y_0 + c + a + b/2$
 $= F(x_0, y_0) + a + b/2$
- But (x_0, y_0) is point on the line and $F(x_0, y_0) = 0$
- Therefore, $d_{start} = a + b/2 = dy - dx/2$
 - * use d_{start} to choose the 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 the decision variable by 2, but does not change the sign
- Bresenham's line algorithm: same but doesn't generalize as nicely to circles and ellipses

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



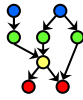
Example Code

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

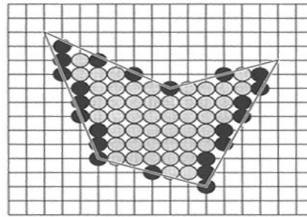
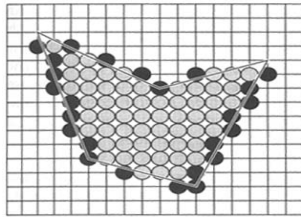
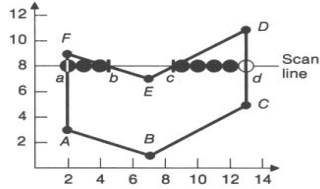
    writePixel(x, y, value);

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

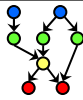
Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Generic Polygons



Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



© 2007 Disney/Pixar



~ Intermission ~
Take A Break!



© 2006 Warner Brothers



© 2001 – 2007 DreamWorks Animation SKG



Polygon Mesh Shading [1]

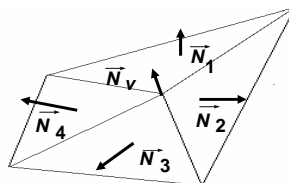
Illumination intensity interpolation

- Gouraud shading
 - use for polygon approximations to curved surfaces
- Linearly interpolate intensity along scan lines
 - eliminates intensity discontinuities at polygon edges; still have gradient discontinuities, mach banding is improved, not eliminated
 - must differentiate desired creases from tessellation artifacts (edges of a cube vs. edges on tessellated sphere)
- Step 1: calculate bogus vertex normals as average of surrounding polygons' normals:

$$\vec{N}_v = \frac{\vec{N}_1 + \vec{N}_2 + \vec{N}_3 + \vec{N}_4}{\|\vec{N}_1 + \vec{N}_2 + \vec{N}_3 + \vec{N}_4\|}$$

More generally:

$$\vec{N}_v = \frac{\sum_{i=1}^n \vec{N}_i}{\left\| \sum_{i=1}^n \vec{N}_i \right\|} \quad n = 3 \text{ or } 4 \text{ usually}$$



- neighboring polygons sharing vertices and edges approximate smoothly curved surfaces and won't have greatly differing surface normals; therefore this approximation is reasonable

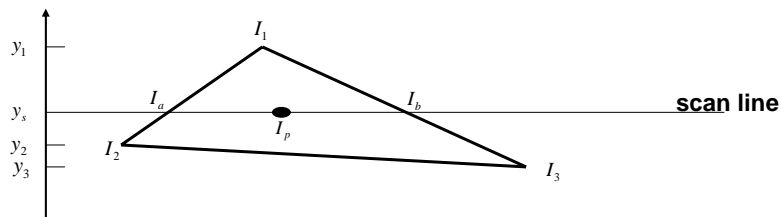
Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Polygon Mesh Shading [2]

Illumination intensity interpolation (cont.)

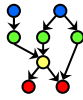
- Step 2: interpolate intensity along polygon edges
- Step 3: interpolate along scan lines



$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2} \quad I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3}$$

$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}$$

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Scan Converting Circles [1]

Version 1: really bad

For $x = -R$ to R

$$y = \sqrt{R \cdot R - x \cdot x};$$

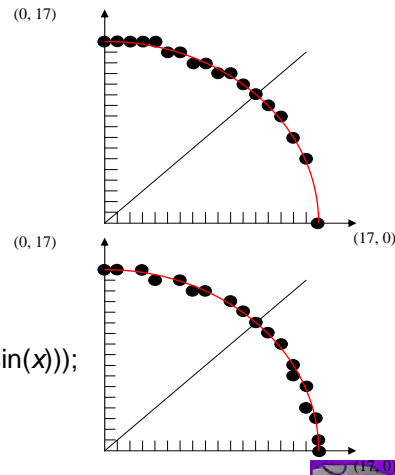
Pixel ($\text{round}(x)$, $\text{round}(y)$);

Pixel ($\text{round}(x)$, $\text{round}(-y)$);

Version 2: slightly less bad

For $x = 0$ to 360

Pixel ($\text{round}(R \cdot \cos(x))$, $\text{round}(R \cdot \sin(x))$);



Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.

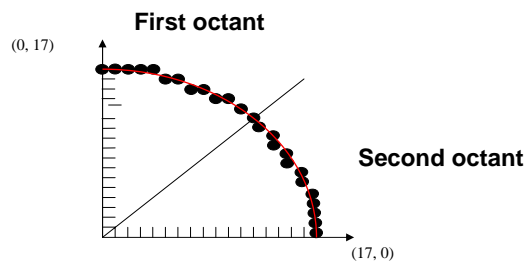


Scan Converting Circles [2]

Version 3: better!

● Midpoint Circle Algorithm:

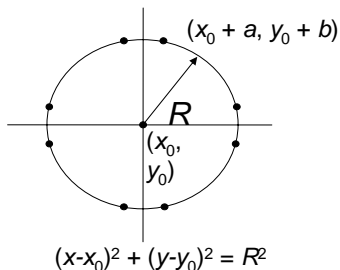
- * First octant generated by algorithm
- * Other octants generated by symmetry



Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Midpoint Circle Algorithm



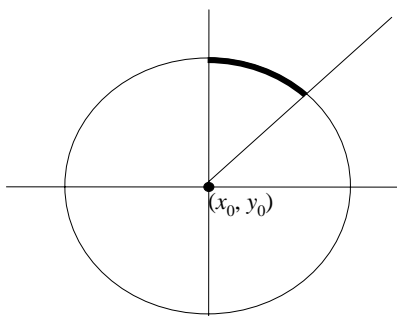
- Symmetry: If $(x_0 + a, y_0 + b)$ is on the circle, so are $(x_0 \pm a, y_0 \pm b)$ and $(x_0 \pm b, y_0 \pm a)$; hence there's an 8-way symmetry.
- BUT, keep in mind that there are always issues when rounding points on a circle to integers

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Using the Symmetry

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



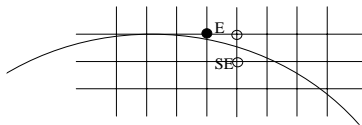
- It starts at $(x_0, y_0 + R)$
- Let's use another incremental algorithm with a decision variable evaluated at midpoint

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.





Sketch of Incremental 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);
}

```

- (*decision_var* will be defined momentarily)
- Note: can replace all occurrences of x_0 and y_0 with 0, 0, Pixel ($x_0 + x$, $y_0 + y$) with Pixel (x , y)
- Essentially a **shift of coordinates**

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



What We Need for Incremental Algorithm

- 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 line strategy: Use the implicit equation of circle

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

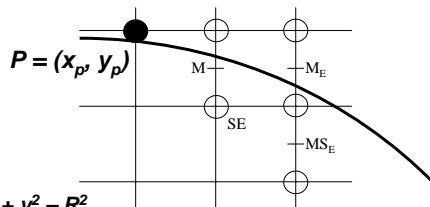
$F(x,y)$ is zero on the circle, negative inside it, positive outside

- If we are at pixel (x, y) , examine $(x + 1, y)$ and $(x + 1, y - 1)$
- Again compute F at the midpoint = $F(\text{midpoint})$

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Decision Variable



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

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

- What we are asking is this: "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?"

- If it is negative there, this midpoint is inside the circle, so the v distance to the circle is less at $(x + 1, y)$ than at $(x + 1, y - 1)$.
- If it is positive, the opposite is true.

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



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 :

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

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.

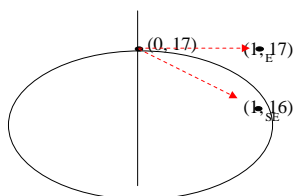


Alternate Specification [1]

- 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

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



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

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

$$F_E - F_{SE} = 290 - 257 = 33$$

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

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Alternate Specification [2]

- 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

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

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Alternate Specification [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.

- The integer $d1$ 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.

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Incremental Computation, Again [1]

- How should we 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 $F(x + 1, y) - F(x, y)$

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

and that

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

is just

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

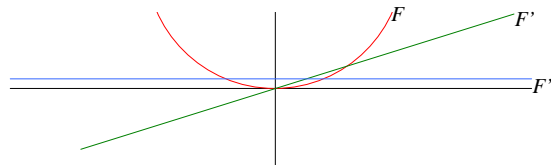
Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.





Incremental Computation, Again [2]

- So if we move E, 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.



Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Second Differences [1]

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

$$D_E(x + 1, y) - D_E(x, y) = 2$$

$$D_E(x + 1, y - 1) - D_E(x, y) = 2$$

- Similarly

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

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

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Second Differences [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 terms as we move.
 - * This is also true of $\Delta_{SE}(x, y)$
- 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 $d(i)$ 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

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



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;
    }
    else /* Move SE */
      y--;
      decision += delta_SE;
      delta_E += 2; delta_SE += 4;
    }
    x++;
    Pixel(x, y);
  }
}

```

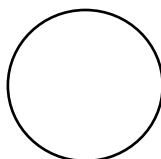
Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.





Analysis

- Uses a float!
- 1 test, 3 or 4 additions per pixel
- Initialization can be improved
- Multiply everything by 4 \Rightarrow No Floats!
 - * This makes the components even, but the sign of the decision variable remains the same



Questions

- Are we getting all pixels whose distance from the circle is less than $\frac{1}{2}$?
- Why is “ $x < y$ ” the right stopping criterion?
- What if it were an ellipse?

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Other Scan Conversion Problems

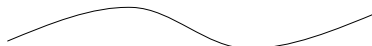
- Patterned primitives



- Aligned Ellipses



- Non-integer primitives



- General conics

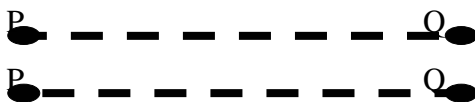


Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Patterned Lines

- Patterned line from P to Q is not 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



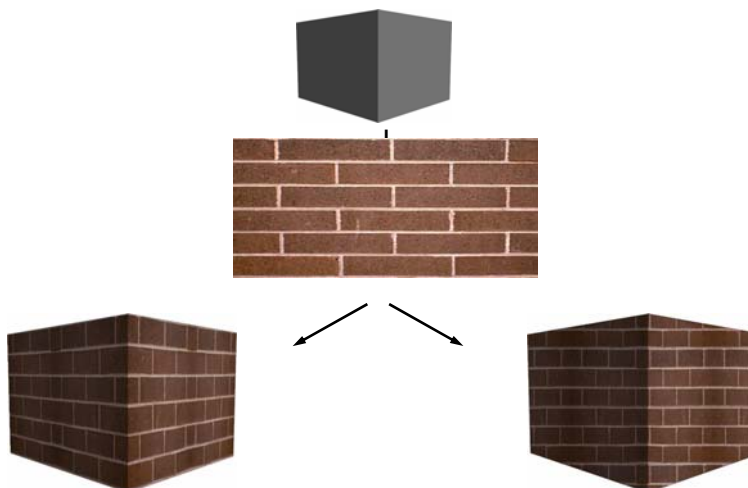
Geometric patterned line



Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Geometric vs. Cosmetic Pattern



Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.





Aligned Ellipses

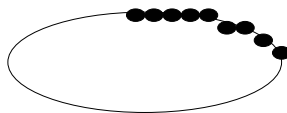
- Equation is

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

i.e.,

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

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

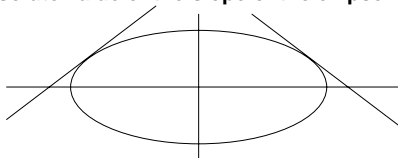


Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Direction-Changing Criterion [1]

- 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

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

- This vector points more right than up when

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

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.





Direction-Changing Criterion [2]

- In our case,

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

and

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

so we check for

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

i.e.

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

- This, too, can be computed incrementally

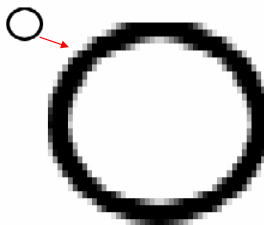
Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Problems with Aligned Ellipses



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



Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Aliasing and Antialiasing

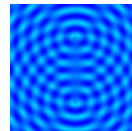
● Definition

- * Effect causing two continuous signals to become indistinguishable (aliases)
- * Occurs when sampled
- * Signals: actual mathematical object (line, polygon, ellipse)
- * General examples: distortion, artifacts

● Specific Examples of Aliasing

- * Spatial aliasing: Moiré pattern (aka Moiré vibration)
- * Jaggies in line, polygon, ellipse scan conversion
- * Temporal aliasing in animation (later)

<http://snurl.com/1yw63>

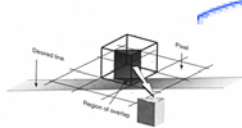


<http://snurl.com/1yw67>

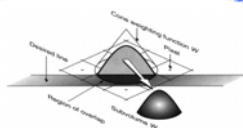


● Antialiasing: Techniques for Prevention

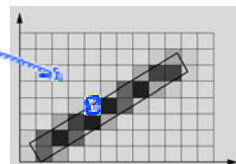
- * (Unweighted) area sampling
- * Pixel weighting: weighted area sampling



Unweighted (box)



Weighted (cone)



<http://snurl.com/1yw5z>



Non-Integer Primitives and General Conics

● 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

● 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 gray-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.

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Clipping Lines

- **Clipping (2.4 Eberly 2e; 3.11-3.12, 6.5.3-6.5.4, FVFH; 7.2-7.6, Angel)**
 - * **Problem**
 - ⇒ Input: coordinates for primitives
 - ⇒ Output: *visible components* of primitives
 - * **Equational solutions: simultaneous, parametric**
 - * **Basic primitive: clip individual points (test against rectangle bounds)**
- **Lines (2.5 Eberly 2e; 3.12, FVD; 7.3, Angel)**
 - * **Clipping line segment AB against viewing rectangle R**
 - * **General idea 1 (equational / regional approach)**
 - ⇒ Divide plane into regions about R, see whether AB can possibly intersect
 - ⇒ Find intersections
 - * **General idea 2 (parametric approach)**
 - ⇒ Express line as parametric equation(s): 1 matrix or 2 scalar
 - ⇒ Find intersections by plugging into parametric equation (Table 3.1, FVFH)
 - ⇒ Use to check clipping cases

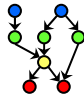


Cohen-Sutherland Algorithm

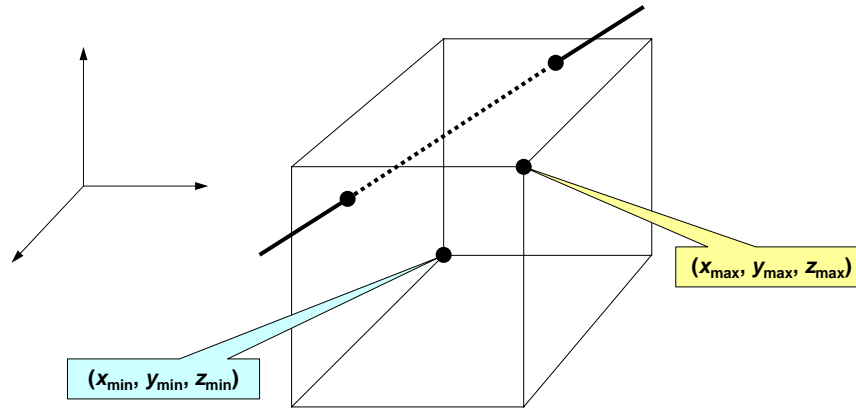
- **General Idea 1 [Cohen and Sutherland, 1963]**
 - * Divide plane into 9 regions about and including R
 - * See whether AB can possibly intersect
- **Outcodes: Quick Rejection Method for Intersection Testing**
 - * **Unique 4-bit binary number for each of 9 regions**

1001	1000	1010	y_{max}
0001	0000	0010	
0101	0100	0110	y_{min}

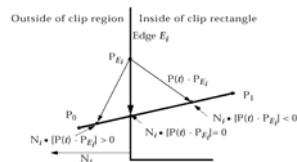
 - ⇒ $b_0 = 1$ iff $y > y_{max}$
 - ⇒ $b_1 = 1$ iff $y < y_{min}$
 - ⇒ $b_2 = 1$ iff $x > x_{max}$
 - ⇒ $b_3 = 1$ iff $x < x_{min}$
 - * **Check clipping cases**
 - ⇒ 8 floating-point subtractions per line segment, plus integer comparison
 - ⇒ Each line segment has 2 outcodes: o_1, o_2
 - ⇒ Case 1: $o_1 = o_2 = 0000$ – inside; show whole segment
 - ⇒ Case 2: $o_1 = 0000, o_2 \neq 0000$ (or vice versa) – partly inside; shorten
 - ⇒ Case 3: $o_1 \& o_2 \neq 0000$ – totally outside; discard
 - ⇒ Case 4: $o_1 \& o_2 = 0000$ – both endpoints outside; check further!



View Volumes in 3D: Perspective Frustum and Parallel Cuboid



Parametric Line Clipping: 2-D (see handout for 3-D)



$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D}$$

Calculations for Parametric Line Clipping Algorithm

Clip Edge _i	Normal N _i	P _{E_i}	P ₀ -P _{E_i}	$t = \frac{N_i \cdot (P_0 - P_{E_i})}{-N_i \cdot D}$
left: x = x _{min}	(-1,0)	(x _{min} , y)	(x ₀ - x _{min} , y ₀ - y)	$\frac{-(x_0 - x_{min})}{(x_1 - x_0)}$
right: x = x _{max}	(1,0)	(x _{max} , y)	(x ₀ - x _{max} , y ₀ - y)	$\frac{-(x_0 - x_{max})}{(x_1 - x_0)}$
bottom: y = y _{min}	(0,-1)	(x, y _{min})	(x ₀ - x, y ₀ - y _{min})	$\frac{-(y_0 - y_{min})}{(y_1 - y_0)}$
top: y = y _{max}	(0,1)	(x, y _{max})	(x ₀ - x, y ₀ - y _{max})	$\frac{-(y_0 - y_{max})}{(y_1 - y_0)}$

Adapted from slides © 2005 A. VanDam, Brown University. Reused with permission.



Summary

- **Scan Conversion**
 - * Lines
 - * Polygons
 - * Circles and Ellipses
- **Methods**
 - * General difference-based approach
 - * Forward differences
 - * Second differences
- **Clipping**
- **Issues**
 - * Minimizing floating-point calculations (and integer calculations!)
 - * Antialiasing
- **Coming Up**
 - * OpenGL Primer (3 parts)
 - * Ray intersection t_{min} with cube, sphere, implicit equations (ray tracing)



Terminology

- **Scan Conversion aka Rasterization**
- **Methods**
 - * Forward differences
 - * Second differences
- **Aliasing**
 - * Effect causing two continuous signals to become indistinguishable (aliases)
 - * Occurs when sampled
 - * Signals: actual mathematical object (line, polygon, ellipse)
 - * General examples: distortion, artifacts
- **Specific Examples of Aliasing**
 - * Spatial aliasing: Moire pattern (aka Moire vibration)
 - * Jaggies in line, polygon, ellipse scan conversion
 - * Temporal aliasing in animation (later)
- **Antialiasing: Techniques for Prevention**
 - * (Unweighted) area sampling
 - * Pixel weighting: weighted area sampling

