Lecture 8 of 41

Scan Conversion 2 of 2: Circles/Ellipses and Polygons

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course pages: http://bit.ly/hGvXlH / http://bit.ly/eVizrE
Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:

Today: Sections 2.4, 2.5 esp. 2.5.4, 3.1.6, Eberly *2e* – see **http://bit.ly/ieUq45**

Next class: Sections 2.5, 2.6.1-2.6.2, 4.3.2, 20.2, Eberly *2e*

Brown CS123: Scan Conversion (**http://bit.ly/hfbF0D**), Shapes (**http://bit.ly/hatPSi**), Polygons/Texture Mapping (**http://bit.ly/h2VZn8**)

Wayback Machine archive of Brown CS123 slides: **http://bit.ly/gAhJbh**

Lecture Outline

- Readings
  - Last class: §2.3.5, 2.4, 3.1.3, Eberly 2e
  - Today's class: §2.4, 2.5 (Especially 2.5.4), 3.1.6, Eberly 2e
  - Next class: §2.5, 2.6.1-2.6.2, 4.3.2, 20.2, Eberly 2e
- Excerpts from Van Dam notes, Brown CS123
  - Scan converting circles/ellipses (starting from 19 in fall, 2010 notes)
  - Polygons (Shapes 2-4)
  - Triangle meshes (Shapes 13-14)
  - Scan line interpolation (Polygons 7; Shading 14, 2005 – 2009 notes)
- Last Time: Intro to Clipping and Culling
  - Clipping: Cohen-Sutherland, Cyrus-Beck / Liang-Barsky
  - Visibility Culling: view frustum, back face, occlusion
- Today: Scan Conversion, Concluded
  - Circles and ellipses
  - Polygons

Where We Are

Lecture	Topic	Primary Source(s)
0	Course Overview	Chapter 1, Eberly 2e
1	**CG Basics: Transformation Matrices; Lab 0**	**Sections (§) 2.1, 2.2**
2	Viewing 1: Overview, Projections	§ 2.2.3 – 2.2.4, 2.8
3	Viewing 2: Viewing Transformation	§ 2.3 esp. 2.3.4; FVFH slides
4	**Lab 1a: Flash & OpenGL Basics**	**Ch. 2, 16[1], Angel *Primer***
5	Viewing 3: Graphics Pipeline	§ 2.3 esp. 2.3.7; 2.6, 2.7
6	Scan Conversion 1: Lines, Midpoint Algorithm	§ 2.5.1, 3.1; FVFH slides
7	**Viewing 4: Clipping & Culling; Lab 1b**	**§ 2.3.5, 2.4, 3.1.3**
8	Scan Conversion 2: Polygons, Clipping Intro	§ 2.4, 2.5 esp. 2.5.4, 3.1.6
9	Surface Detail 1: Illumination & Shading	§ 2.5, 2.6.1 – 2.6.2, 4.3.2, 20.2
10	**Lab 2a: Direct3D / DirectX Intro**	**§ 2.7, Direct3D handout**
11	Surface Detail 2: Textures; OpenGL Shading	§ 2.6.3, 20.3 – 20.4, *Primer*
12	Surface Detail 3: Mappings; OpenGL Textures	§ 20.5 – 20.13
13	**Surface Detail 4: Pixel/Vertex Shad.; Lab 2b**	**§ 3.1**
14	Surface Detail 5: Direct3D Shading; OGLSL	§ 3.2 – 3.4, Direct3D handout
15	Demos 1: CGA, Fun; Scene Graphs: State	§ 4.1 – 4.3, **CGA handout**
16	**Lab 3a: Shading & Transparency**	**§ 2.6, 20.1, *Primer***
17	**Animation 1: Basics, Keyframes; HW/Exam**	**§ 5.1 – 5.2**
	Exam 1 review; Hour Exam 1 (evening)	**Chapters 1 – 4, 20**
18	**Scene Graphs: Rendering; Lab 3b: Shader**	**§ 4.4 – 4.7**
19	**Demos 2: SFX; Skinning, Morphing**	**§ 5.3 – 5.5, CGA handout**
20	Demos 3: Surfaces; B-reps/Volume Graphics	§ 10.4, 12.7, **Mesh handout**

Lightly-shaded entries denote the due date of a written problem set; heavily-shaded entries, that of a machine problem (programming assignment); blue-shaded entries, that of a paper review; and the green-shaded entry, that of the term project.

Green, **blue** and **red** letters denote **exam review**, **exam**, and **exam solution review** dates.

Drawing Circles, Versions1 & 2

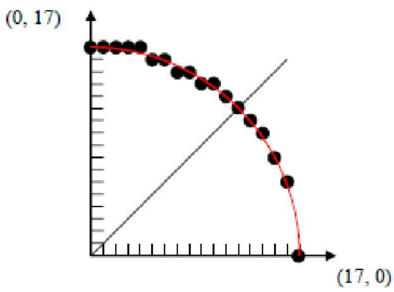
Version 1: <u>really</u> bad

For x from $-R$ *to* R:

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

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

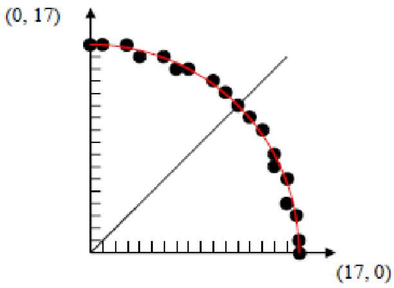
Pixel (round(x), round($-y$));



Version 2: slightly less bad

For x from 0 *to* 360:

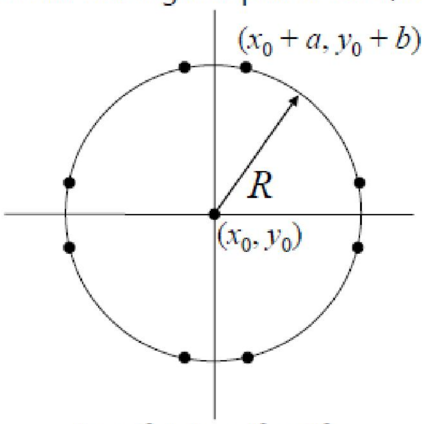
Pixel (round ($R * \cos(x)$), round ($R * \sin(x)$));



Drawing Circles, Version 3

- 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

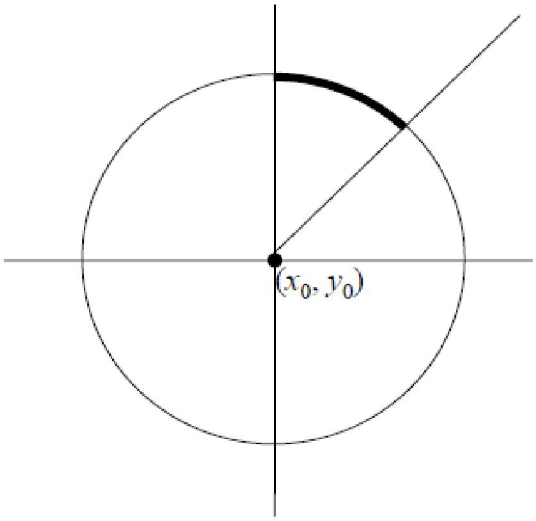


$$(x-x_0)^2 + (y-y_0)^2 = R^2$$

Adapted from slides © 1997 – 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

Using The Symmetry

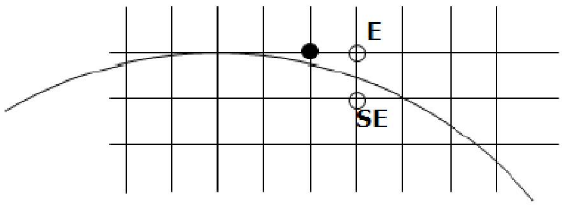
- Scan top right 1/8 of circle of radius R
- Circle starts at $(x_0, y_0 + R)$
- Let's use another incremental algorithm with decision variable evaluated at midpoint



7

Incremental Algorithm [1]: Sketch

```
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 variable
    }
    else {
        /* move south east */
        update decision variable
        y--;
    }
    Pixel(x, y);
}
```



- Note: can replace all occurrences of x_0, y_0 with 0, shifting coordinates by $(-x_0, -y_0)$

Incremental Algorithm [2]: Computations needed

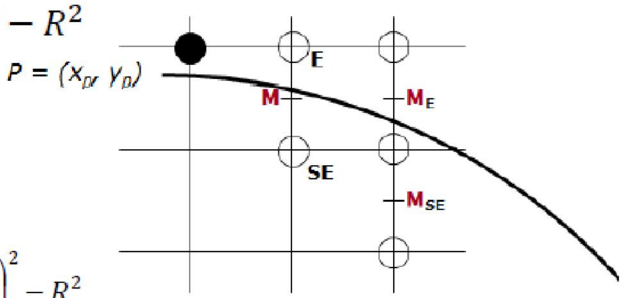
- Decision variable
  - negative if we move E, positive if we move SE (or vice versa).

- 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**

Right Decision Variable?

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

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

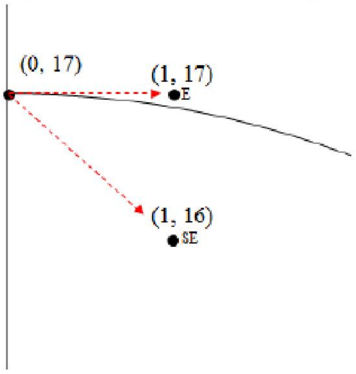
$$d\big((x+1, y-1), Circ\big) = \sqrt{(x+1)^2 + (y-1)^2} - R$$

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

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

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



$$f_E = 1^2 + 17^2 = 290$$
$$f_{SE} = 1^2 + 16^2 = 257$$

$$f_E - f_{SE} = 290 - 257 = 33$$
$$2y - 1 = 2(17) - 1 = 33$$

Alternate Phrasing [2]

- The second value, which is always less, is *closer* if its difference from R^2 is less than: $\frac{1}{2}(2y - 1)$

i.e., if $\quad R^2 - [(x+1)^2 + (y-1)^2] < \frac{1}{2}(2y-1)$

then

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

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

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

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

Alternate Phrasing [3]

- The ***radial distance decision*** is whether

$$d_1 = (x+1)^2 + \left(y - \frac{1}{2}\right)^2 + \frac{1}{4} - R^2$$

is positive or negative.

- The ***vertical distance decision*** is whether

$$d_2 = (x+1)^2 + \left(y - \frac{1}{2}\right)^2 - R^2$$

is positive or negative; d_1 and d_2 are ¼ apart.

- The integer d_1 is positive only if $d_2 +$ ¼ is positive (except special case where $d_2 = 0$: remember you're using integers).

Incremental Algorithm Revisited [1]

- How to compute the value of

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

at successive points? (vertical distance approach)

- Answer: Note that $f(x+1,y) - f(x,y)$

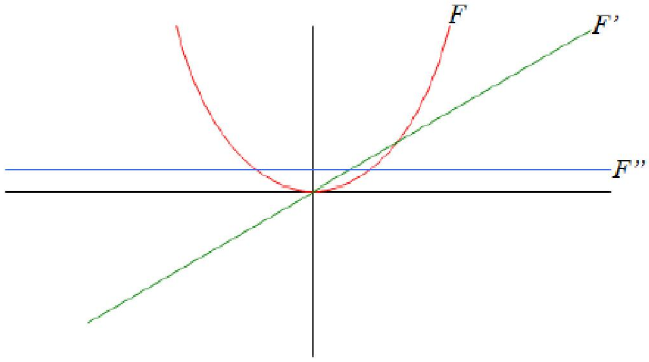
  is $\Delta_E(x,y) = 2x + 3$

  and that $f(x+1,y-1) - f(x,y)$

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

Incremental Algorithm Revisited [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]

- The function $\Delta_{\mathrm{E}}(x, y) = 2x + 3$ is linear, hence amenable to incremental computation:

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

- Similarly

$$\Delta_{\mathrm{SE}}(x+1, y) - \Delta_{\mathrm{SE}}(x, y) = 2$$
$$\Delta_{\mathrm{SE}}(x+1, y-1) - \Delta_{\mathrm{SE}}(x, y) = 4$$

Second Differences [2]

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

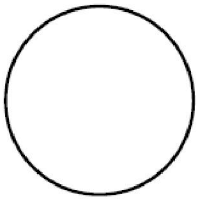
Midpoint Eighth Circle Algorithm

```
MEC (R) /* 1/8th of a circle w/ radius R */ {
    int     x = 0, y = R;
    int     delta_E     = 2*x + 3;
    int     delta_SE    = 2(x-y) + 5;
    float   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); } }
```

Adapted from slides © 1997 – 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f   Reused with permission.

Analysis

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



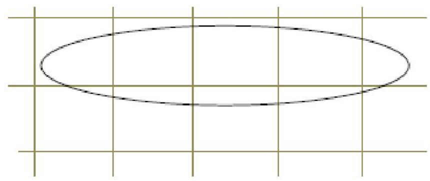
Questions

- 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?

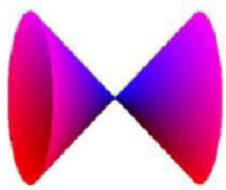
Other Scan Conversion Problems

- Aligned Ellipses

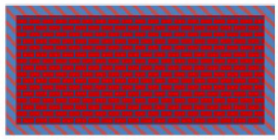
- Non-integer primitives



- General conics

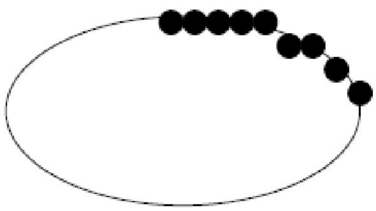


- Patterned primitives



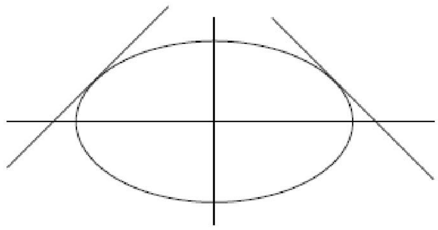
Aligned Ellipses

- 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]

- 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 $f(x, y)$ which is

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

- 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]

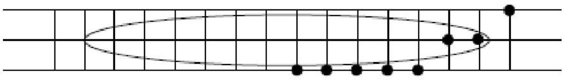
- In our case, $\frac{\partial f}{\partial x}(x, y) = 2a^2x$ and $\frac{\partial f}{\partial y}(x, y) = 2b^2y$

  so we check for

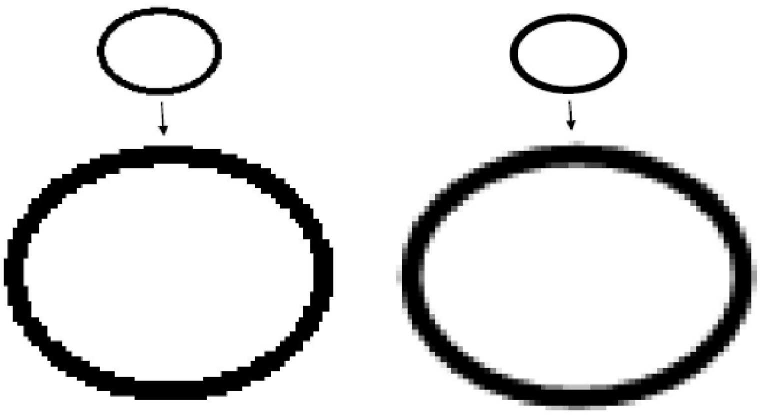
$$2a^2x - 2b^2y > 0$$
$$a^2x - b^2y > 0$$

- This, too, can be computed incrementally

Problems with Aligned Ellipses

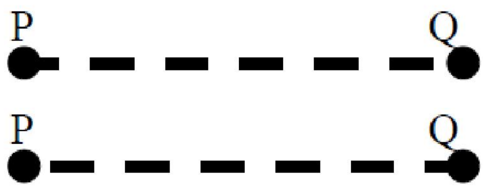


- Now in ENE octant, not ESE octant
- This problem is an artifact of *aliasing*, remember filter?



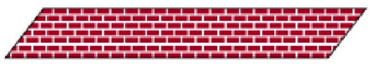
Patterned Lines

- Patterned line from *P* to *Q* is not same as patterned line from *Q* to *P*.

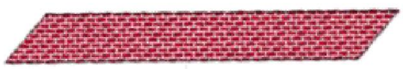


- Patterns can be ***cosmetic*** or ***geometric***
  - Cosmetic: Texture applied after transformations
  - Geometric: Pattern subject to transformations

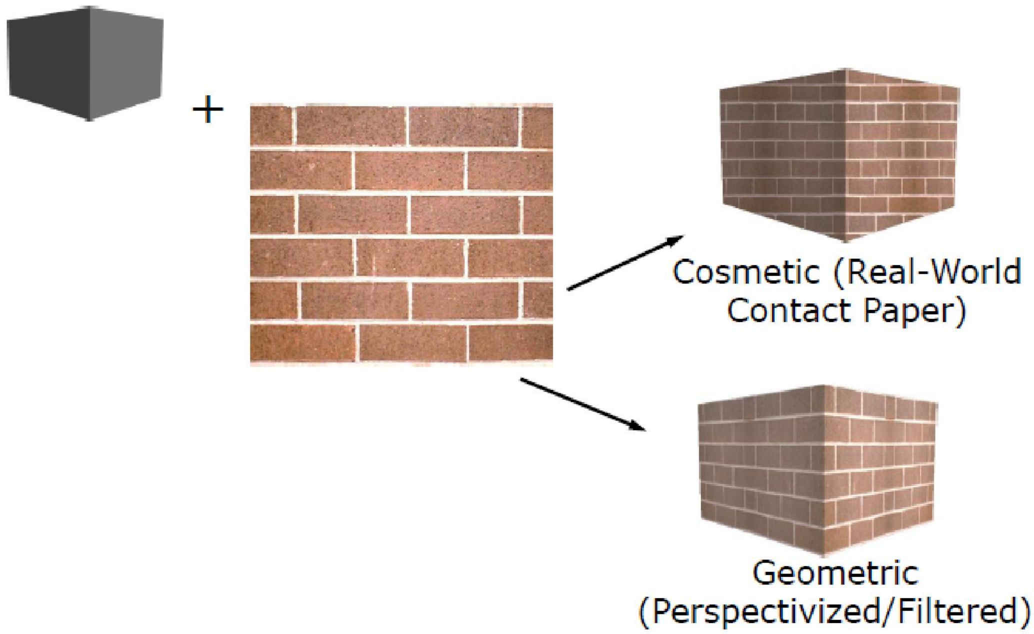
Cosmetic patterned line



Geometric patterned line



Geometric vs. Cosmetic



Adapted from slides © 1997 – 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

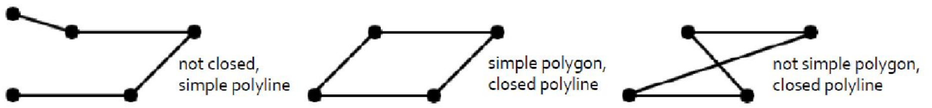
Non-Integer Primitives & General Conics

- **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 drawing gray-scale conics is easier than drawing B/W conics

2-D Object Definition [1]

- Lines and polylines:
  - Polylines: lines drawn between ordered points
  - A closed polyline is a polygon, a simple polygon has no self-intersections

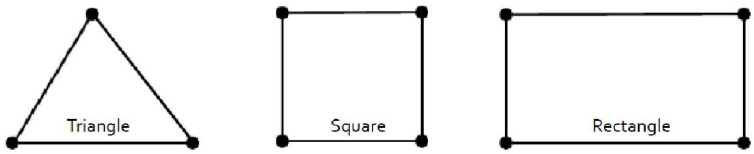


- Convex and concave polygons:
  - Convex: For every pair of points inside the polygon, the line between them is entirely inside the polygon.
  - Concave: For some pair of points inside the polygon, the line between them is not entirely inside the polygon. Not Convex.

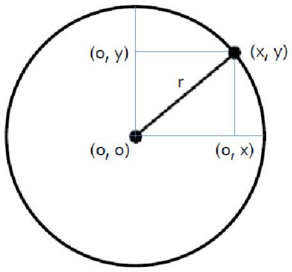
Brown University CS123
Shapes
Slide 2 (fall, 2010)
http://bit.ly/h2VZn8

2-D Object Definition [2]

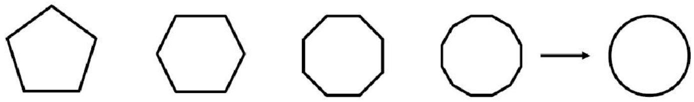
- Special Polygons:



- Circles:
  - Set of all points equidistant from one point called the center
  - The distance from the center is the radius *r*
  - The equation for a circle centered at (0, 0) is $r^2 = x^2 + y^2$



  - A circle can be approximated by a polygon with many sides.



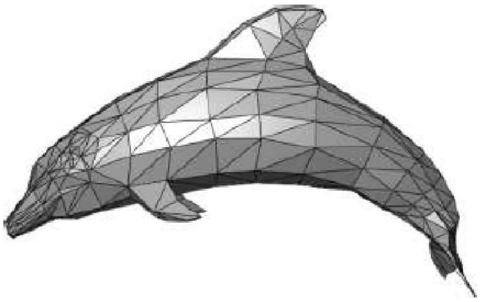
Brown University CS123
Shapes
Slides 3-4 (fall, 2010)
http://bit.ly/h2VZn8

Triangle Meshes

- Most common representation of shape in three dimensions
- All vertices of triangle are guaranteed to lie in one plane (not true for quadrilaterals or other polygons)
- Uniformity makes it easy to perform mesh operations such as subdivision, simplification, transformation etc.
- Many different ways to represent triangular meshes

Brown University CS123
Shapes
Slide 13 (fall, 2010)
http://bit.ly/h2VZn8

- See chapters 9 and 28 in book, en.wikipdia.org/wiki/polygon_mesh
  - Mesh transformation and deformation
  - Procedural generation techniques (upcoming labs on simulating terrain)



Adapted from slides © 1997 – 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

Triangular Mesh Representation

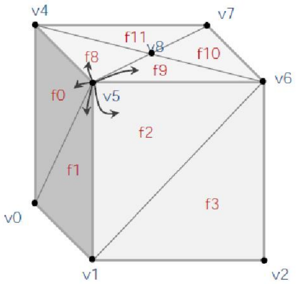
- Vertex and face tables, analogous to 2D vertex and edge tables
- Each vertex listed once, triangles listed as ordered triplets of indices into the vertex table
  - Edges inferred from triangles
  - It's often useful to store associated faces with vertices (i.e. computing normals: vertex normal average of surrounding face normals)
- Vertices listed in counter clockwise order in face table.
  - No longer just because of convention. CCW order differentiates front and back of face

Vertex List

v0	0, 0, 0	f0 f1 f12 f15 f7
v1	1, 0, 0	f2 f3 f13 f12 f1
v2	1, 1, 0	f4 f5 f14 f13 f3
v3	0, 1, 0	f6 f7 f15 f14 f5
v4	0, 0, 1	f6 f7 f0 f8 f11
v5	1, 0, 1	f0 f1 f2 f9 f8
v6	1, 1, 1	f2 f3 f4 f10 f9
v7	0, 1, 1	f4 f5 f6 f11 f10
v8	.5, .5, 0	f8 f9 f10 f11
v9	.5, .5, 1	f12 f13 f14 f15

Face List

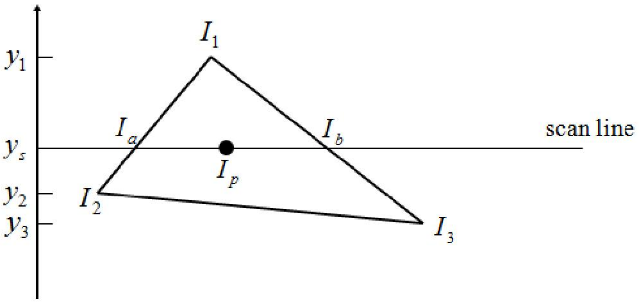
f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7



Brown University CS123
Shapes
Slide 14 (fall, 2010)
http://bit.ly/h2VZn8

General Polygons [1]: Scan Line Interpolation

- **1. Interpolate Value Along Polygon Edges to Get I_a, I_b**
- **2. Interpolate Value Along Scan Lines to Get I_p**



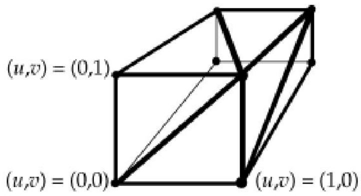
$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3}$$

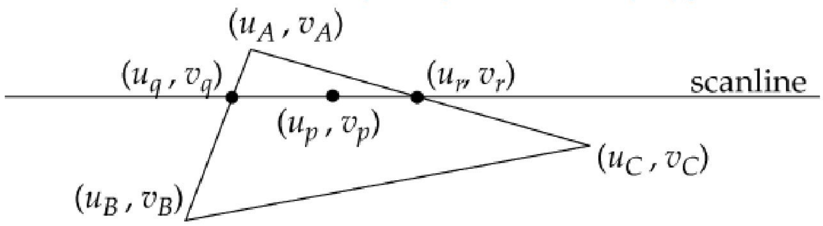
$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}$$

General Polygons [2]: Texture Mapping Preview

- Texture mapping polygons
  - (u, v) texture coordinates are pre-calculated and specified per vertex
  - Vertices may have different texture coordinates for different faces



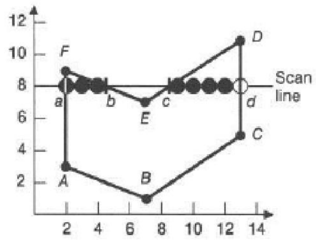
  - Texture coordinates are linearly interpolated across polygon

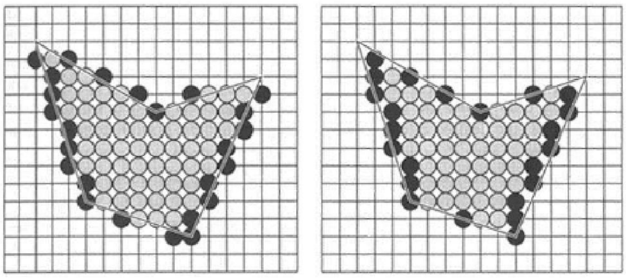


Brown University CS123
(Polygons &) Texture Mapping
Slide 7 (fall, 2010)
http://bit.ly/h2VZn8

Adapted from slides © 1997 – 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

General Polygons [3]: Continuity and Scan Line Interpolation





Brown University CS123
Scan Conversion
Slide 43 (fall, 2010)
http://bit.ly/hfbF0D

- what's the difference between these two solutions? Under which circumstances is the right one "better"?

Summary

- **Last Time: Clipping and Culling**
  - What parts of scene to clip: edges *vs.* polygons of model
  - What parts of viewport to clip against: clip faces *vs.* clip edges
  - Cohen-Sutherland clipping: outcodes, simultaneous equations
  - Liang-Barsky / Cyrus-Beck clipping: parametric equations
  - Visibility culling: view frustum, back face, occlusion
- **Today: Scan Conversion, Concluded**
  - Circles and ellipses
  - Polygons: scan line interpolation (for flat/constant shading)
  - Later: Gouraud & Phong shading, z-buffering, texture mapping
- **Excerpts from Van Dam notes, Brown CS123**
  - Scan converting circles/ellipses
  - Polygons
  - Triangle meshes
  - Scan line interpolation

Terminology

- **Scan Conversion (*aka* Rasterization)**
  - Given: geometric object (*e.g.*, circle, ellipse, projected polygon)
  - Decide: what pixels to light (turn on; later, color/shade)
  - Basis: what part of pixels crossed by object
- **Issues (Reasons why Scan Conversion is Nontrivial Problem)**
  - Aliasing (*e.g.*, jaggies) – discontinuities in lines
  - Cracks: discontinuities in "polygon" mesh
- **Drawing Circles & Ellipses**
  - Incremental algorithm – uses rounding, floating point arithmetic
  - Forward differences – precalculated amounts to add to running total
  - Decision variable – value whose sign indicates which pixel is next
- **Drawing Polygons**
  - Texture mapping – finding pixels of image (texture) to put in polygon
  - Scan line interpolation – procedure for filling in closed curves