Lecture 8 of 41

Scan Conversion 2 of 2: Circles/Ellipses and Polygons

William H. Hsu
Department of Computing and Information Sciences, KSU

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

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

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Topic</th>
<th>Primary Source(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Course Overview</td>
<td>Chapter 1, Eberly Z'</td>
</tr>
<tr>
<td>1</td>
<td>CG Basics: Transformation Matrices; Lab 0</td>
<td>Sections 6.1, 2.1, 2.2</td>
</tr>
<tr>
<td>2</td>
<td>Viewing 1: Viewing Projections</td>
<td>§ 2.3 esp. 2.3.4, FFP slides</td>
</tr>
<tr>
<td>3</td>
<td>Viewing 2: Viewing Transformation</td>
<td>§ 2.3 esp. 2.3.4, FFP slides</td>
</tr>
<tr>
<td>4</td>
<td>Lab 1: Flash &amp; OpenGL Basics</td>
<td>Ch. 2, 16, Angel Primer</td>
</tr>
<tr>
<td>5</td>
<td>Viewing 3: Graphics Pipeline</td>
<td>§ 2.3 esp. 2.3.7, 2.6.7</td>
</tr>
<tr>
<td>6</td>
<td>Scan Conversion 1: Lines, Midpoint Algorithm</td>
<td>§ 2.5.1, 3.1, FFP slides</td>
</tr>
<tr>
<td>7</td>
<td>Viewing 4: Clipping &amp; Clipping: Lab 1b</td>
<td>§ 2.3.6, 2.4, 3.1.1</td>
</tr>
<tr>
<td>8</td>
<td>Scan Conversion 2: Polygon, Clipping Intro</td>
<td>§ 2.4, 3.6 esp. 2.4, 3.1.1</td>
</tr>
<tr>
<td>9</td>
<td>Surface Detail 1: Illumination &amp; Shading</td>
<td>§ 2.5, 2.6.1 – 2.6.2, 4.3.2, 20.2</td>
</tr>
<tr>
<td>10</td>
<td>Lab 2a: Direct3D / DirectX Intro</td>
<td>§ 2.7, Direct3D handout</td>
</tr>
<tr>
<td>11</td>
<td>Surface Detail 2: Textures, OpenGL Shading</td>
<td>§ 2.6.3, 20.4 – 20.4. Primer</td>
</tr>
<tr>
<td>12</td>
<td>Surface Detail 3: Mappings; OpenGL Textures</td>
<td>§ 2.5 – 20.13</td>
</tr>
<tr>
<td>13</td>
<td>Surface Detail 4: Pixel/Vertex Shad.: Lab 2b</td>
<td>§ 3.1</td>
</tr>
<tr>
<td>14</td>
<td>Surface Detail 5: Direct3D Shading; CGLSL</td>
<td>§ 3.2 – 3.4, Direct3D handout</td>
</tr>
<tr>
<td>15</td>
<td>Demos 1: CGA, Fun, Scene Graphs, State</td>
<td>§ 4.1 – 4.3, CGA handout</td>
</tr>
<tr>
<td>16</td>
<td>Lab 3a: Shading &amp; Transparency</td>
<td>§ 2.5.1, Primer</td>
</tr>
<tr>
<td>17</td>
<td>Animation 1: Basics, Keyframes; HW/Exam</td>
<td>§ 6.1 – 6.2</td>
</tr>
<tr>
<td>18</td>
<td>Scene Graphs: Rendering; Lab 3b: Shader</td>
<td>§ 6.6 – 8.6, CGA handout</td>
</tr>
<tr>
<td>19</td>
<td>Demos 2: SFX, Skinning, Morphing</td>
<td>Exam 1 review, Hour Exam 1 review (evening)</td>
</tr>
<tr>
<td>20</td>
<td>Demos 3: Surfaces, B-reps/Volume Graphics</td>
<td>Ch. 1 – 4, 20</td>
</tr>
</tbody>
</table>

Lectures

- Lecture 8 of 41

Drawing Circles, Versions 1 & 2

**Version 1: really bad**

For \( x \) from \(-R\) to \(R\):

\[
y = \sqrt{R^2 - x^2};
\]

Pixel \( (\text{round}(x), \text{round}(y)) \):

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

**Version 2: slightly less bad**

For \( x \) from 0 to 360:

Pixel \( (\text{round}(R \cos(x)), \text{round}(R \sin(x))) \):

Adapted from slides © 1997 – 2010 van Dam et al., Brown University

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

\[
(x-x_0)^2 + (y-y_0)^2 = R^2
\]

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
Incremental Algorithm [1]: Sketch

\[ x = x_0, \ y = y_0 + R, \ \text{Pixel}(x, y); \]

\[ \text{for} \ [(x = x_0 + 1); \ (x - x_0) > (y - y_0); \ x++ \} \{ \]

\[ \text{if} \ (\text{decision var} < 0) \} \{
    \text{/* move east */}
    \text{update decision variable}
\}

\[ \text{else} \{ \text{/* move south east */}
    \text{update decision variable}
    y--; \}
\]

\[ \text{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

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Decision Variable

- Evaluate \( f(x, y) = x^2 + y^2 - R^2 \)
  at the point: \( (x + 1, y - \frac{1}{2}) \)
  
- 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 \( d_{\text{vert}} \) are proportional
- For circles, not true:
  \[
  d((x + 1, y), \text{Circ}) = \sqrt{(x + 1)^2 + y^2} - R
  \]
  \[
  d((x + 1, y - 1), \text{Circ}) = \sqrt{(x + 1)^2 + (y - 1)^2} - R
  \]
  
- Which \( d \) is closer to zero? (i.e. which of the two values below is closer to \( R \)):
  \[
  \sqrt{(x + 1)^2 + y^2} \text{ or } \sqrt{(x + 1)^2 + (y - 1)^2}
  \]

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Alternate Phrasing [1]

- We could ask instead:
  
  \[ (x + 1)^2 + y^2 \text{ or } (x + 1)^2 + (y - 1)^2 \text{ closer to } R^2? \]

- The two values in equation above differ by

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

\[
\begin{align*}
(0, 17) & \quad (1, 17) \\
(1, 17) & \quad (1, 16)
\end{align*}
\]

\[
\begin{align*}
\ell_1 &= 1^2 + 17^2 = 290 \\
\ell_2 &= 1^2 + 16^2 = 257 \\
\ell_2 - \ell_1 &= 290 - 257 = 33 \\
2y - 1 &= 2(17) - 1 = 33
\end{align*}
\]

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

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

then

\[
\begin{align*}
0 &< y - \frac{1}{2} + (x + 1)^2 + (y - 1)^2 - R^2 \\
0 &< (x + 1)^2 + y^2 - 2 y + 1 + y - \frac{1}{2} - R^2 \\
0 &< (x + 1)^2 + y^2 - y + \frac{1}{2} - R^2 \\
0 &< (x + 1)^2 + (y - \frac{1}{2})^2 + \frac{1}{4} - R^2
\end{align*}
\]
**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 \( \frac{1}{4} \) apart.

- The integer \( d_1 \) is positive only if \( d_2 + \frac{1}{4} \) 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 \(\Delta_E(x, y) = 2x + 3\) is linear, hence amenable to incremental computation:
  \[
  \begin{align*}
  \Delta_E(x+1, y) - \Delta_E(x, y) &= 2 \\
  \Delta_E(x+1, y-1) - \Delta_E(x, y) &= 2
  \end{align*}
  \]
  
  - Similarly
  \[
  \begin{align*}
  \Delta_{SE}(x+1, y) - \Delta_{SE}(x, y) &= 2 \\
  \Delta_{SE}(x+1, y-1) - \Delta_{SE}(x, y) &= 4
  \end{align*}
  \]
Second Differences [2]

- For any step, 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 $d$ 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

Adapted from slides © 1997 – 2010 van Dam et al., Brown University

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
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 all 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 \( \Delta_E \) and \( \Delta_{SE} \) 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

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

- 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]**

- 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 the same as patterned line from $Q$ to $P$.
- Patterns can be cosmetic or geometric:
  - Cosmetic: Texture applied after transformations
  - Geometric: Pattern subject to transformations

Geometric vs. Cosmetic

- Cosmetic (Real-World Contact Paper)
- Geometric (Perspectivized/Filtered)
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.
2-D Object Definition [2]

- Special Polygons:
  - Triangle
  - Square
  - Rectangle

- 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 \((a, b)\) is \( r^2 = x^2 + y^2 \)
  - A circle can be approximated by a polygon with many sides.

Adapted from slides © 1997 – 2010 van Dam et al., Brown University

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

- See chapters 9 and 28 in book,
en.wikipedia.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
Triangular Mesh Representation

- Vertex and face tables, analogous to 1D 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

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 \( I_p \)
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

\[ (u_A, v_A) \]
\[ (u_B, v_B) \]
\[ (u_C, v_C) \]
\[ (u_D, v_D) \]

- Texture coordinates are linearly interpolated across polygon

... scanline ...

Adapted from slides © 1997 – 2010 van Dam et al., Brown University

---

General Polygons [3]:
Continuity and Scan Line Interpolation

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

... diagram ...

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
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