who's who

end user

application program

graphics pipeline

who's who for today

user

graphics pipeline

graphics pipeline

user's description of scene

1. Build scene
2. Projection transform
3. Clip
4. Perspective division
5. Viewport transform
6. Scan convert

Vertex Transformations

Algorithm
graphics pipeline overview

• simplified pipeline
• general pipeline

projection

perspective projection

orthographic projection
(viewer at infinity)

standard orthographic view volume

view volume is 2x2x2, axis-aligned, and centered at the origin

rendered image is the parallel projection of scene onto front plane

simplified graphics pipeline

user's description of scene in standardized, homogenous world coordinates

clip

eliminate "outside" primitive
viewport transformation

standardized world coordinates

display coordinates

viewport transformation

standardized world coordinates

display coordinates

scan conversion

display coordinates

frame buffer

simplified graphics pipeline

user's description of scene in standardized homogenous world coordinates

clip

Viewport transform

Scan convert

frame buffer coordinates

pixel (0,0) sampled at (½, ½)
frame buffer coordinates: alternative

pixel (0,0) sampled at (0,0)

we won’t use this though

frame buffer coordinates

today: a pixel is a little square!!

pixel "on"

pixel "off"

scan conversion

- points
- line segments
- polygons

scan conversion: point

closest pixel

(scan converting line segments)

- naive algorithm
- midpoint algorithm
- bresenham’s algorithm
1-pixel wide lines

one pixel per row

one pixel per column

scan conversion

- input: endpoint coordinates
- output: pixels to turn on (and their color) for a 1-pixel wide line segment

endpoints: \((\tfrac{1}{2}, \tfrac{1}{2})\) and \((9\tfrac{1}{2}, 4\tfrac{1}{2})\)

\[y = mx + b, \quad m = \frac{4}{9}, \quad b = \frac{5}{18}\]

for each column \(i\), compute the y-intercept at \(x = i + \frac{1}{2}\)

\(\lfloor m \star (i + \frac{1}{2}) + b \rfloor\)

naïve algorithm

- input: endpoints \((x_0, y_0)\) to \((x_1, y_1)\)
  
  for now we'll assume \(x_0 < x_1\)
  
  line: \(y = mx + b\) where \(m = \frac{y_1 - y_0}{x_1 - x_0}, \quad b = y_0 - mx_0\)

  for \(i = \lfloor x_0 \rfloor \ldots \lfloor x_1 \rfloor\)
  
  turn on pixel \((i, \lfloor m(i + \frac{1}{2}) + b \rfloor)\)

  is there a better/faster algorithm? yes, we can avoid (almost all) multiplication!

midpoint algorithm: \(0 \leq m \leq 1\)

- suppose we've just turned on pixel \((i, j)\)
- next we'll turn on
  
  NE: \((i+1, j+1)\) or
  
  E: \((i+1, j)\)
Suppose we've just turned on pixel \((i, j)\).

Next we'll turn on
- **NE**: if the \(y\) intercept at \(x = i + 3/2\) is at least \(j + 1\).

For example, \(m > 1\):

\[
\begin{align*}
\text{NE: if } j + 1 &\leq m(i + 3/2) + b \\
\text{E: otherwise}
\end{align*}
\]

### Advantage of Midpoint Algorithm

If the endpoints of the line segment have integer coordinates we can avoid floating point operations.

```
 this was a big deal in the dark ages!
```
bresenham’s algorithm (0 ≤ m ≤ 1)

1. turn on \((i, j)\) where \(i = \lfloor x_0 \rfloor\) \(j = \lfloor y_0 \rfloor\)
   
   //find first pixel

bresenham’s algorithm (0 ≤ m ≤ 1)

2. \(d = 2\Delta_x (j+1) - \Delta_y (2i+3) - 2\Delta_x b\)
   // initialize d

bresenham’s algorithm (0 ≤ m ≤ 1)

3. while \(i ≤ x_1\) {
   if \(d ≤ 0\) // go NE {
   } else // go E {
   }
}

bresenham’s algorithm (0 ≤ m ≤ 1)

3. while \(i ≤ x_1\) {
   if \(d ≤ 0\) // go NE {
   } else // go E {
   }
}

bresenham’s algorithm (0 ≤ m ≤ 1)

3. while \(i ≤ x_1\) {
   if \(d ≤ 0\) // go NE {
   } else // go E {
   }
}

bresenham’s algorithm (0 ≤ m ≤ 1)

3. while \(i ≤ x_1\) {
   if \(d ≤ 0\) // go NE {
   } else // go E {
   }
}

bresenham’s algorithm (0 ≤ m ≤ 1)

3. while \(i ≤ x_1\) {
   if \(d ≤ 0\) // go NE {
   } else // go E {
   }
}

bresenham’s algorithm: other cases

- similar rules

e.g. \(m > 1\)
scan conversion

- input: endpoint coordinates
- output: pixels to turn on for a 1-pixel wide line segment + pixel color

shading models

- What is the color here?
- Color is defined at vertices!!!!!!!

flat shading

- use color of first vertex

smooth (gouraud) shading

- interpolate

interpolation computation

- \( c + \frac{\Delta c}{\Delta x} \)

scan conversion

- points
- line segments
- polygons
polygon: \( v_0, v_1, v_2, v_3, v_4 \)

\[ \begin{array}{c}
\hat{v}_1 \\
\hat{v}_2 \\
\hat{v}_3 \\
\hat{v}_4 \\
\hat{v}_0 \\
\end{array} \]

polygon: scan conversion

\[
polygon(v_0, \ldots, v_{n-1}) \\
\text{for } i=0 \text{ to } n-1 \\
\text{draw-line-segment}(p_i, p_{i+1 \mod n})
\]

scan conversion

- points
- line segments
- polygons
  - filled

scan conversion

- input: vertex coordinates
- output: pixels to turn on (and their color) for filled polygon

which pixels should be on?

here we get the same size!
For each scan line:
1. Find edge/scan line intersection points
2. Order by x-coordinate
3. Use odd-even test to turn on pixels

Odd-even test:
- Crossing edge changes in/out state
- Note: For polygon, we'll always have an even number of edge crossings

Tessellating polygons:

Odd-even test:
- We count this as one crossing!

Odd-even test:
- Buyer beware!
Scan Line Algorithm

1. Find edge/scan line intersection points
2. Order by x-coordinate
3. Use odd-even test to turn on pixels

Scan Line Notation

Scan line i is defined as the line $y = i + \frac{1}{2}$

Edge Notation

$(x_{\text{bottom}}, y_{\text{bottom}})$
$(x_{\text{top}}, y_{\text{top}})$

$y_{\text{bottom}} \leq y_{\text{top}}$

Naïve Algorithm

For each edge of polygon:
- Compute intersection of current scan line & edge line
- Check if intersection is on edge segment

Scan Line Algorithm

For each scan line:
1. Find edge/scan line intersection points
2. Order by x-coordinate
3. Use odd-even test to turn on pixels

We want to do this FAST
exploit coherence

let \( E_i \) be the edges that intersect scan line \( i \) then \\
\[ E_i = E_{i-1} + \text{edges with } i - \frac{1}{2} \leq y_{\text{bottom}} < i + \frac{1}{2} \]
\[ - \text{edges with } y_{\text{top}} \leq i + \frac{1}{2} \]

data structures

- edge table
  - for each scan line \( i \) a list of edges with \\
  \[ i - \frac{1}{2} < y_{\text{bottom}} \leq i + \frac{1}{2} \]
- active edge list
  - edges intersecting current scan line

edge table (et)

<table>
<thead>
<tr>
<th>scan line</th>
<th>edge list</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

list of edges with with:
\[ 2\frac{1}{2} < y_{\text{bottom}} \leq 3\frac{1}{2} \]

exercise: edge table

data structures

- edge table
  - for each scan line \( i \) a list of edges with \\
  \[ i - \frac{1}{2} < y_{\text{bottom}} \leq i + \frac{1}{2} \]
- active edge list
  - edges intersecting current scan line

example edge table
example active edge list

ael update

ael changes

scan line algorithm

intersection point calculation

edge record at scan line i

- $y_{\text{bottom}}$
- $y_{\text{top}}$
- $1/m$
- $x_{\text{int}}$ : x-intercept at scan line $i$

initialize to x-intercept at first scan line the edge intersects

for each scan line
1. find edge/scan line intersection points
2. order by x-coordinate
3. use odd-even test to turn on pixels

we want to do this FAST

end/begin two end two begin

ael always contains an even number of edges!

all we need to store is x-intercept, $X$, and inverse slope

$y = mx + b$

scan line $i$: $y = i + \frac{1}{2}$

scan line $i-1$: $y = i - \frac{1}{2}$

$(X, i - \frac{1}{2})$

$(X, i + \frac{1}{2})$
exercise: initialize edge records

<table>
<thead>
<tr>
<th>edge</th>
<th>( \gamma_{\text{bottom}} )</th>
<th>( y_{\text{top}} )</th>
<th>( x_{\text{int}} )</th>
<th>1/m</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_0</td>
<td>1</td>
<td>7</td>
<td>( \frac{7}{6} )</td>
<td>( \frac{1}{3} )</td>
</tr>
<tr>
<td>e_1</td>
<td>1</td>
<td>3</td>
<td>( \frac{3}{2} )</td>
<td>0</td>
</tr>
<tr>
<td>e_2</td>
<td>1</td>
<td>6</td>
<td>( \frac{5}{2} )</td>
<td>( \frac{1}{3} )</td>
</tr>
<tr>
<td>e_3</td>
<td>3</td>
<td>6</td>
<td>( \frac{1}{3} )</td>
<td>0</td>
</tr>
<tr>
<td>e_4</td>
<td>3</td>
<td>6</td>
<td>( \frac{1}{3} )</td>
<td>0</td>
</tr>
</tbody>
</table>

initialize edge records

scan line algorithm

```plaintext
build et
scanLine = -1
ael = φ
while scanLine < h
    scanLine ++
    for each edge in ael:  \( x_{\text{int}} += \frac{1}{m} \)
    ael += et[scanLine]
    ael -= {edges with \( y_{\text{top}} \leq \) scanLine + \( \frac{1}{2} \)}
    sort edges in ael by \( x_{\text{int}} \)
    compute "on" pixels by odd-even rule
```

scan line algorithm

```plaintext
//compute "on" pixels by odd-even rule
for k = 0 ... aelNumEdges/2
    for j + \( \frac{1}{2} \) > aelEdges[2k]\( x_{\text{int}} \) and
    j + \( \frac{1}{2} \) \leq aelEdges[2k+1]\( x_{\text{int}} \)
        turn on pixels (j, scanLine)
```

initialize edge records
initialize edge table

<table>
<thead>
<tr>
<th>scan line</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>e₂, e₃</td>
</tr>
<tr>
<td>3</td>
<td>e₄</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>e₀, e₁</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

example

\[ ael = \emptyset \]

scan line = -1

example

\[ ael = \emptyset \]

scan line = 0

example

\[ ael \text{ sorted by } x_{int} \]

scan line = 1

example

\[ ael \text{ sorted by } x_{int} \]

scan line = 2

example

\[ ael \text{ sorted by } x_{int} \]

scan line = 3
example
ael sorted by x_{int}
<table>
<thead>
<tr>
<th>edge</th>
<th>x_{int}</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_0</td>
<td>3</td>
</tr>
<tr>
<td>e_1</td>
<td>10/4</td>
</tr>
<tr>
<td>e_2</td>
<td>6</td>
</tr>
</tbody>
</table>

scanLine = 4

example
ael sorted by x_{int}
<table>
<thead>
<tr>
<th>edge</th>
<th>x_{int}</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_0</td>
<td>23/8</td>
</tr>
<tr>
<td>e_1</td>
<td>5</td>
</tr>
<tr>
<td>e_2</td>
<td>22/4</td>
</tr>
<tr>
<td>e_3</td>
<td>6</td>
</tr>
</tbody>
</table>

scanLine = 5

example
ael sorted by x_{int}
<table>
<thead>
<tr>
<th>edge</th>
<th>x_{int}</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_0</td>
<td>17/6</td>
</tr>
<tr>
<td>e_1</td>
<td>1</td>
</tr>
</tbody>
</table>

scanLine = 6

example
ael sorted by x_{int}
<table>
<thead>
<tr>
<th>edge</th>
<th>x_{int}</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_0</td>
<td>1</td>
</tr>
</tbody>
</table>

scanLine = 7

tessellation: center claims
tie breaker 1: left owns
tie breaker 2: above owns

exercise

- where in the algorithm are these tie-breaking rules specified?

horizontal edges

what is in aet when scanLine = 2

Exercise

scan conversion

- input: vertex coordinates
- output: pixels to turn on for filled polygon + pixel color

shading models

Color is defined at vertices!!!!!!
flat shading

Color entire polygon the color of first vertex.

smooth shading

1. Interpolate along edges.

what happens here?

edge record at scan line $i$

- $y_{\text{bottom}}$
- $y_{\text{top}}$
- $1/m$
- $x_{\text{int}}$
- $c_{\text{int}}$
- $\Delta c / \Delta x$

scan conversion

- points
- lines segments
- polygons
  - filled

the frame and $z$ buffers and hidden surface removal
display coordinates vs. frame buffer

display coordinates

(0,0)

frame buffer

fb(h-j-1)

(0,0)

pixel (i,j)

scan conversion

- points
- lines segments
- polygons
  - filled

hidden surface removal

which is right?

hidden surface removal

compare z-depth of corresponding points on 3d surfaces

z-depth

What is the z-depth here?

z-depth is defined at vertices!!!!!! so interpolate

z-depth

1. Interpolate along edges.
2. Interpolate along scan line.
edge record at scan line i

- $y_{\text{top}}$
- $x_i$
- $1/m$
- $c_i$
- $\Delta_i$
- $z_i$
- $\delta_i$

depth on edge at $(i,x_i)$

depth increment: to compute $z_i$ incrementally

z-buffering

- Frame Buffer
- Z-Buffer

initialize the z-buffer

\[
\begin{array}{ccc}
-1 & -1 & -1 \\
-1 & -1 & -1 \\
-1 & -1 & -1 \\
\end{array}
\]

scan conversion

- Without z-buffering:

  \[
  \text{fb}(i,j) = \text{currcolor};
  \]

- With z-buffering

  // $z$ val is the normalized depth of the point on the polygon
  // that projects to point $(i,j)$

  Compute $z$ val

  If $z$ val > $zb(i,j)$
      \[
      \text{fb}(i,j) = \text{currcolor}
      \]
      \[
      zb(i,j) = z$\text{val}$
      \]

graphics pipeline overview

- simplified pipeline
- general pipeline

  - geometric primitives defined in standardized, homogenous world coordinates
  - orthographic projection
  - standardized view volume
  - standard viewport transformation