1. Build scene
2. Clip
3. Project
4. Scan convert
Done

1. Build scene
2. Clip
3. Project
4. Scan convert

3. Project
projected 2d vertices & their 3d depth
2d image

projected 2d vertex

othographic
perspective
orthographic projection

- 3d coordinates \((x,y,z)\)
- 3d projected coordinates
- 2d projected coordinates
- depth

orthographic projection

4. Scan convert
3. Project
3d vertices
2d image

normalized device coordinates

\(-1,1\)
\((1,-1)\)

normalized device coordinates

\(-1,1\)
\((1,-1)\)
normalize – step 1

(normalize - step 1)

normalize – step 2

(normalize - step 2)

normalize – step 3

(normalize - step 3)

orthographic projection matrix

(orthographic projection matrix)

now for perspective projection...

(now for perspective projection...)

perspective view volume

(perspective view volume)
Perspective projection:

\[(x,y,z) \rightarrow (x \cdot z_{near}/z, y \cdot z_{near}/z, z_{near})\]

X-projection:

\[x'/x = z_{near}/z\]

Depth-dependent scale:

\[z = z_0 \rightarrow z = z_1\]

Depth dependent x,y scale:

1. Depth dependent scale
   \[(x,y,z) \rightarrow (x \cdot z_{near}/z, y \cdot z_{near}/z, z_{near})\]

2. Orthographic projection
perspective projection matrix

depth-dependent scale is not a 3D linear transform

BUT we can fake it

\[
\begin{pmatrix}
  n & 0 & 0 & 0 \\
  0 & n & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix}
=
\begin{pmatrix}
  nx/z \\
  ny/z \\
  m \\
  1
\end{pmatrix}
\]

view coordinates
normalize by w component

11/5/2002  CS155 - Subject  97

perspective projection matrix

depth-dependent scale is not a 3D linear transform

BUT we can fake it

\[
\begin{pmatrix}
  nx/z \\
  ny/z \\
  m \\
  1
\end{pmatrix}
\]

there are many hacks to fix this ... next we'll show the one we will use!!

11/5/2002  CS155 - Subject  98

depth dependent x,y scale

relative depth info is all we need

11/5/2002  CS155 - Subject  99

perspective projection

\[
\begin{pmatrix}
  2/(r-l) & 0 & 0 & 0 \\
  0 & 2/(t-b) & 0 & 0 \\
  0 & 0 & 2/(f-n) & -(f+n)/(f-n) \\
  0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x' \\
  y' \\
  z'/w' \\
  w'
\end{pmatrix}
=
\begin{pmatrix}
  x'/w' \\
  y'/w' \\
  z'/w' \\
  1
\end{pmatrix}
\]

view coordinates
normalization

11/5/2002  CS155 - Subject  100

projection

\[
\begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix}
\rightarrow
\begin{pmatrix}
  x' \\
  y' \\
  z'/w' \\
  w'
\end{pmatrix}
\rightarrow
\begin{pmatrix}
  x'/w' \\
  y'/w' \\
  1
\end{pmatrix}
\]

view coordinates
normalized device coordinates
2D projection (x'/w',y',w')
relative depth z'/w'

11/5/2002  CS155 - Subject  101

projection matrix: \( M_p \)

\[
\begin{pmatrix}
  2/(r-l) & 0 & 0 & 0 \\
  0 & 2/(t-b) & 0 & 0 \\
  0 & 0 & 2/(f-n) & -(f+n)/(f-n) \\
  0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x' \\
  y' \\
  z'/w' \\
  w'
\end{pmatrix}
=
\begin{pmatrix}
  x'/w' \\
  y'/w' \\
  z'/w' \\
  1
\end{pmatrix}
\]

perspective orthographic

11/5/2002  CS155 - Subject  102
geometric primitives

object coordinates: \( v \)  
world coordinates: \( M_wv \)  
view coordinates: \( M_vM_wv \)  
normalized device coordinates: \( M_pM_vM_wv \)

description of vertex
situated in world
as seen from a particular viewpoint
seen from viewpoint in a normalized world

graphics pipeline

1. Build scene
2. Clip
3. Project
4. Scan convert

Done

scan conversion

2d normalized device coordinates

display window

screen window can be re-sized by end user

viewport transformation

view window

display window: set by end-user
aspect ratio may be different
viewport transformation

display window

view window

viewport transformation

display window

our default transformation

display window: set by end-user

view window

our viewport transform

2d normalized device coordinates

(1,1)

(0,0)

(-1,1)

(1,-1)

(0,-1)

2D translate & scale

display coordinates

(w,h)

(0,h)

(0,0)

(w,0)

display coordinates

(0,0)

sampled at (0,0)

(-½, -½)

(w-½, h-½)

(w-½,-½)

we won't use this though

display coordinates

(w,h)

(0,h)

(w,0)

pixel "on"

pixel location

pixel "off"

today: a pixel is a little square!!

point on image plane

pixel location

(point on image plane

**scan conversion**

- points
- line segments
- polygons

**scan conversion: point**

- point
- nearest neighbors
- closest pixel

**line segments**

- 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
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 \(m = (y_1 - y_0)/(x_1 - x_0), b = y_0 - mx_0\)
  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: if the y-intercept at \(x = i + 3/2\) is at least \(j+1\)
  - E: otherwise

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

- 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 \(j+1\)
  - E: otherwise
midpoint algorithm: $0 \leq m \leq 1$

- suppose we’ve just turned on pixel $(i,j)$
- next we’ll turn on
  - NE: if $j+1 \leq m(i+3/2) + b$
  - E: otherwise

midpoint algorithm: other cases

- similar rules

E: $\text{NE}$

e.g. $m > 1$

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!

avoiding (almost all) multiplication ($0 \leq m \leq 1$)

- we just turned on pixel $(i,j)$
- next we’ll turn on

  - NE: if $j+1 \leq m(i+3/2) + b$
  - E: otherwise

  $j+1 \leq m(i+3/2) + b \leftrightarrow \Delta_x(j+1) < \Delta_y(i+3/2) \cdot \Delta_x b$

  where $\Delta_x = x_1 - x_0$ and $\Delta_y = y_1 - y_0$

  $\leftrightarrow 2\Delta_x(j+1) \leq \Delta_y (2i+3) + 2\Delta_x b$

  $\leftrightarrow \Delta_x(j+1) - \Delta_y (2i+3) - 2\Delta_x b \leq 0$

we can compute $d$ incrementally without multiplication

\[ d \]

bresenham’s algorithm ($0 \leq m \leq 1$)

1. turn on $(i,j)$ where $i = \lfloor x_0 \rfloor$, $j = \lfloor y_0 \rfloor$

   \[ (x_0, y_0) \]

   // find first pixel

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

   // initialize $d$
bresenham's algorithm \( \{0 \leq m \leq 1\}\)

3. while \( i \leq x_1 \) {
   
   if \( d \leq 0 \)     // go NE
   {
      +++ turn on pixel \((i,j)\)
      update \( d 
      \)
   }
   
   else  // go E
   {
      +++ turn on pixel \((i,j)\)
      update \( d 
      \)
   }

   ++
   
   turn on pixel \((i,j)\)

   \( i++, j++ \)

   \( d = d + 2 \Delta x - 2 \Delta y \)

   ++

   turn on pixel \((i,j)\)

   \( i++ \)

   \( d = d - 2 \Delta y \)

bresenham's algorithm: other cases

• similar rules

\( e.g. \, m>1 \)

\[ \begin{array}{c|c}
N & \text{NE} \\
\hline
\end{array} \]

\( \begin{array}{c|c}
N & \text{NE} \\
\hline
\end{array} \)

scan conversion

• input: endpoint coordinates

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

shading models

\text{What is the color here?}

\text{Color is defined at vertices!!!!!!}
**flat shading**

What is the color here?

use color of first vertex

---

**smooth (gouraud) shading**

What is the color here?

interpolate

---

**interpolation computation**

\[ c + \frac{\Delta c}{\Delta x} \]

---

**scan conversion**

- points
- line segments
- polygons

---

**polygon: \( v_0, v_1, v_2, v_3, v_4 \)**

\[ \text{polygon}(v_0, \ldots, v_{n-1}) \]

for \( i = 0 \) to \( n-1 \)

\[ \text{draw-line-segment}(p_i, p_{i+1 \mod n}) \]
scan conversion

- points
- line segments
- polygons
  - filled

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!

tesselating polygons

tesselating polygons with edges

exercise: mark remaining edge pixels
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

odd-even test

crossing edge changes in/out state
note: for polygon we'll always have an even number of edge crossings

odd-even test

we count this as one crossing!

odd-even test

buyer beware!

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

use your favorite sorting algorithm
**Scan Line Algorithm**

1. 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.

**Scan Line Notation**

- Scan line 0
- Scan line 1
- Scan line h-1

**Edge Notation**

- $(x_{bottom}, y_{bottom})$
- $(x_{top}, y_{top})$

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

**Naïve Algorithm**

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

**Scan Line Algorithm**

- We want to do this FAST.

**Exploit Coherence**

- Let $E_i$ be the edges that intersect scan line $i$.
- Then $E_i = E_{i-1} +$ edges with $i - \frac{1}{2} < y_{bottom}$.
- $y_{top} \leq i + \frac{1}{2}$.
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 $2\frac{1}{2} < y_{\text{bottom}} \leq 3\frac{1}{2}$

example edge table

<table>
<thead>
<tr>
<th>scan line</th>
<th>edge list</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>e_2, e_5</td>
</tr>
<tr>
<td>3</td>
<td>e_6</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>e_0, e_1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</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 active edge list

ael update

$ael = ael + et[i] - \text{edges with } y_{\text{top}} \leq i + \frac{1}{2}$
**ael changes**

- end/begin
- two end
- two begin

ael always contains an even number of edges!

---

**Intersection point calculation**

- **Scan line i:** \( y = i + \frac{1}{2} \)
- **Scan line i-1:** \( y = i - \frac{1}{2} \)

\( y = mx + b \)

all we need to store is x-intercept

---

**Initialize edge records**

<table>
<thead>
<tr>
<th>edge</th>
<th>( y_{\text{bottom}} )</th>
<th>( y_{\text{top}} )</th>
<th>( \frac{1}{m} )</th>
<th>( x_{\text{int}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_0</td>
<td>1</td>
<td>3</td>
<td>1/3</td>
<td>0</td>
</tr>
<tr>
<td>e_1</td>
<td>1</td>
<td>4</td>
<td>1/4</td>
<td>0</td>
</tr>
<tr>
<td>e_2</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e_3</td>
<td>4</td>
<td>10</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>e_4</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Scan line algorithm**

- **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 \( y_{\text{top}} \leq \frac{\text{scanLine}}{2} \)}
- sort edges in ael by \( x_{\text{int}} \)
- compute "on" pixels by odd-even rule
scan line algorithm

build et
scanLine=-1
ael=∅
while scanLine < h
    scanLine ++
    for each edge in ael:  \[ x_{int} += \frac{1}{m} \]
    ael += et[scanLine]
    ael -= {edges with \( y_{top} \leq \text{scanLine} + \frac{1}{2} \)}
    sort edges in ael by \( x_{int} \)

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

initialize edge records

<table>
<thead>
<tr>
<th>edge</th>
<th>x_{bottom}</th>
<th>x_{top}</th>
<th>y_{top}</th>
<th>x_{int}</th>
<th>\text{ln}</th>
</tr>
</thead>
<tbody>
<tr>
<td>e0</td>
<td>1</td>
<td>7</td>
<td>7/6</td>
<td>1/3</td>
<td>-0</td>
</tr>
<tr>
<td>e1</td>
<td>1</td>
<td>3</td>
<td>6/4</td>
<td>5/2</td>
<td>-2</td>
</tr>
<tr>
<td>e2</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>-5</td>
</tr>
<tr>
<td>e3</td>
<td>4</td>
<td>8</td>
<td>15/4</td>
<td>3/2</td>
<td>-6</td>
</tr>
<tr>
<td>e4</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>-6</td>
</tr>
</tbody>
</table>

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>( e_2,e_3 )</td>
</tr>
<tr>
<td>3</td>
<td>( e_4 )</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>( e_0,e_1 )</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

example

ael = ∅
example

ael sorted by $x_{int}$

<table>
<thead>
<tr>
<th>edge $x_{int}$</th>
<th>e0</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
<th>e4</th>
</tr>
</thead>
<tbody>
<tr>
<td>e0</td>
<td>9/6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e1</td>
<td>9/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

scanLine = 1

example

ael sorted by $x_{int}$

<table>
<thead>
<tr>
<th>edge $x_{int}$</th>
<th>e0</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
<th>e4</th>
</tr>
</thead>
<tbody>
<tr>
<td>e0</td>
<td>9/6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e1</td>
<td>19/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

scanLine = 2

example

ael sorted by $x_{int}$

<table>
<thead>
<tr>
<th>edge $x_{int}$</th>
<th>e0</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
<th>e4</th>
</tr>
</thead>
<tbody>
<tr>
<td>e0</td>
<td>12/6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

scanLine = 3

example

ael sorted by $x_{int}$

<table>
<thead>
<tr>
<th>edge $x_{int}$</th>
<th>e0</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
<th>e4</th>
</tr>
</thead>
<tbody>
<tr>
<td>e0</td>
<td>15/6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e2</td>
<td>15/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e3</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

scanLine = 5

example

ael sorted by $x_{int}$

<table>
<thead>
<tr>
<th>edge $x_{int}$</th>
<th>e0</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
<th>e4</th>
</tr>
</thead>
<tbody>
<tr>
<td>e0</td>
<td>17/6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

scanLine = 6
example

ael sorted by xint

edge \( x_{\text{int}} \)

\[ \text{scanLine} = 7 \]

\[ e_0 \]
\[ e_1 \]
\[ e_2 \]
\[ e_3 \]
\[ e_4 \]


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
Exercise

scan conversion

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

shading models

What is the color here?

Color is defined at vertices!!!!!!

flat shading

Color entire polygon the color of first vertex.

smooth shading

1. Interpolate along edges.

smooth shading

1. Interpolate along edges.
2. Interpolate along scan line between edges.
**what happens here?**

![Diagram of a star]

**edge record at scan line i**

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

color on edge at $(x_{\text{int}}, i)$

color increment

---

**scan conversion**

- points
- lines segments
- polygons
  - filled

the frame and $z$ buffers and hidden surface removal

**display coordinates vs. frame buffer**

- display coordinates
- frame buffer

Pixel $(i, j)$

$fb(i, j)$

---

**hidden surface removal**

- points
- lines segments
- polygons
  - filled

the frame & $z$ buffers and 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

ez-depth

1. Interpolate along edges.
2. Interpolate along scan line.

edge record at scan line i

- \( y_{\text{top}} \)
- \( x_i \)
- \( 1/m \)
- \( c_i \)
- \( \Delta_i \)
- \( z_i \) ← depth on edge at \((i,x_i)\)
- \( \delta_i \) ← depth increment; to compute \( z_i \) incrementally

z-buffering

Frame Buffer

Z-Buffer

Color of surface at point p

Depth of surface at point p

initialize the z-buffer

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

- Without z-buffering:
  \( fb(i,h-j) = \text{currcolor} \)

- 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 > \(zb(i,h-j)\) {
    \( fb(i,h-j) = \text{currcolor} \)
    \( zb(i,h-j) = z\)val
  }

graphics pipeline

1. Build scene
2. Clip
3. Project
4. Scan convert
Done