Filling Polygons
Filling Polygons

Convex

Non-Convex
Filling Polygons
Filling Convex Polygons

activeEdgeList = null;
for (y=yLow to y=yHigh)
{
    activeEdgeList = updateActiveEdgeList(y);
    (xL, xR) = FindxSpan(y, activeEdgeList);
    for (x=xL to x = xR) setPixel(x, y);
}

y = yHigh

y = yLow
Potential problems

1. Special cases like passing through a vertex.
Filling Convex Polygons

Potential problems

1. Special cases like passing through a vertex.

2. Work correctly with polygons sharing edges (no gaps).
Filling Convex Polygons

Potential problems

1. Special cases like passing through a vertex.

2. Work correctly with polygons sharing edges (no gaps).
Filling Convex Polygons

Potential problems

1. Special cases like passing through a vertex.

2. Work correctly with polygons sharing edges (no gaps).

3. Find x spans efficiently.
Filling Convex Polygons

Classify edges of polygon:

If vertices are specified clockwise with \( y_{n+1} = y_0 \)

\[
\{ \\
\quad \text{if } (y_{i+1} - y_i) > 0 \text{ classify entering; } \\
\quad \text{else} \\
\quad \quad \text{if } (y_{i+1} - y_i) < 0 \text{ classify exiting; } \\
\quad \text{else ignore;} \\
\}
\]
Computing scanline/edge intersections.

Suppose $xf$ is the true floating point value of the intersection of (non-horizontal) edge $e$ with integer scanline $y$. 
Computing scanline/edge intersections.

Suppose $x_f$ is the true floating point value of the intersection of (non-horizontal) edge $e$ with integer scanline $y$.

Case 1. $x_f$ not an integer and $e$ is entering. Round $x_f$ up to $x$. 
Computing scanline/edge intersections.

Suppose $xf$ is the true floating point value of the intersection of (non-horizontal) edge $e$ with integer scanline $y$.

Case 1. $xf$ not an integer and $e$ is entering. Round $xf$ up to $x$.
Case 2. $xf$ not an integer and $e$ is exiting. Round $xf$ down to $x$.
Computing scanline/edge intersections.

Suppose $x_f$ is the true floating point value of the intersection of (non-horizontal) edge $e$ with integer scanline $y$.

Case 1. $x_f$ not an integer and $e$ is entering. Round $x_f$ up to $x$.
Case 2. $x_f$ not an integer and $e$ is exiting. Round $x_f$ down to $x$.
Case 3. $x_f$ is an integer, $(x_f,y)$ not a vertex and $e$ entering. $x = x_f$. 

Outside

Inside
Computing scanline/edge intersections.

Suppose \(xf\) is the true floating point value of the intersection of (non-horizontal) edge \(e\) with integer scanline \(y\).

Case 1. \(xf\) not an integer and \(e\) is entering. Round \(xf\) up to \(x\).
Case 2. \(xf\) not an integer and \(e\) is exiting. Round \(xf\) down to \(x\).
Case 3. \(xf\) is an integer, \((xf,y)\) not a vertex and \(e\) entering. \(x = xf\).
Case 4. \(xf\) is an integer, \((xf,y)\) not a vertex and \(e\) exiting. \(x = xf-1\).
Computing scanline/edge intersections.

Suppose $xf$ is the true floating point value of the intersection of (non-horizontal) edge $e$ with integer scanline $y$.

Case 1. $xf$ not an integer and $e$ is entering. Round $xf$ up to $x$.
Case 2. $xf$ not an integer and $e$ is exiting. Round $xf$ down to $x$.
Case 3. $xf$ is an integer, $(xf,y)$ not a vertex and $e$ entering. $x = xf$.
Case 4. $xf$ is an integer, $(xf,y)$ not a vertex and $e$ exiting. $x = xf - 1$.
Case 5. $(xf,y)$ is a vertex and $y = y_{\text{min}}$ for $e$. $x = xf$. 

\[
\begin{array}{c}
  \text{Diagram of edge intersections with scanline.} \\
  \end{array}
\]
Computing scanline/edge intersections.

Suppose $xf$ is the true floating point value of the intersection of (non-horizontal) edge $e$ with integer scanline $y$.

Case 1. $xf$ not an integer and $e$ is entering. Round $xf$ up to $x$.
Case 2. $xf$ not an integer and $e$ is exiting. Round $xf$ down to $x$.
Case 3. $xf$ is an integer, $(xf,y)$ not a vertex and $e$ entering. $x = xf$.
Case 4. $xf$ is an integer, $(xf,y)$ not a vertex and $e$ exiting. $x = xf-1$.
Case 5. $(xf,y)$ is a vertex and $y = ymin$ for $e$. $x = xf$.
Case 6. $(xf,y)$ is a vertex and $y = ymax$ for $e$. Ignore.
Efficient Computation of scanline/edge intersection
Efficient Computation of scanline/edge intersection

Since $y$ is always incremented, we decide whether to increment or decrement $x$

\[ y = y_{\text{High}} \]

\[ y = y_{\text{Low}} \]
(xInit, yInit) and (xTerm, yTerm) determined by clockwise ordering.

yLow = min(yInit, yTerm);
yHigh = max(yInit, yTerm);

dy = yTerm - yInit;
dx = xTerm - xInit;
x = xLow;
for (y = yLow to yHigh)
    x = x + dx/dy;
\( (x_{\text{Init}}, y_{\text{Init}}) \) and \( (x_{\text{Term}}, y_{\text{Term}}) \) determined by clockwise ordering.

\[
\begin{align*}
y_{\text{Low}} &= \min(y_{\text{Init}}, y_{\text{Term}}); \\
y_{\text{High}} &= \max(y_{\text{Init}}, y_{\text{Term}}); \\
dy &= y_{\text{Term}} - y_{\text{Init}}; \\
dx &= x_{\text{Term}} - x_{\text{Init}}; \\
x &= x_{\text{Low}}; \\
\text{for} \, (y = y_{\text{Low}} \text{ to } y_{\text{High}}) \\
&\quad x = x + dx/dy;
\end{align*}
\]

\[
dx/dy = k + F/dy, \text{ where } k \text{ an integer and } |F| < |dy|
\]
x0 = xLow

Update once:
\[ x1 = x0 + \frac{dx}{dy} = x0 + k0 + \frac{F0}{dy} \]
\[ = \text{int}X0 + \frac{F0}{dy} \]

Update again:
\[ x2 = x1 + \frac{dx}{dy} = \text{int}X0 + \frac{F0}{dy} + \frac{dx}{dy} \]
\[ = \text{int}X0 + \frac{(F0+dx)}{dy} \]
\[ = \text{int}X0 + k1 + \frac{F1}{dy} \]
\[ = \text{int}X1 + \frac{F1}{dy} \]
x0 = xLow

Update once:
   x1 = x0 + dx/dy = x0 + k0 + F0/dy
       = intX0 + F0/dy

Update again:
   x2 = x1 + dx/dy = intX0 + F0/dy + dx/dy
       = intX0 + (F0+dx)/dy
       = intX0 + k1 + F1/dy
       = intX1 + F1/dy

Edge Data Structure

| xLow | yLow | xHigh | yHigh | dx | dy | F  | intX | x  |
Edge Data Structure

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
x_{\text{Low}} & y_{\text{Low}} & x_{\text{High}} & y_{\text{High}} & dx & dy & F & \text{IntX} \\
\hline
\end{array}
\]

Update:

\[
F = F + dx;
\]
while (\(|F| \geq |dy|\))

\{
    \text{if } F = F + dy \text{ makes } |F| \text{ smaller}
    \{
        F = F + dy;
        \text{IntX}--; \\
    \}
    \text{else}
    \{
        F = F - dy;
        \text{IntX}++; \\
    \}
\}
4 Update Cases

1. Entering edge (dy > 0) and dx > 0.

2. Entering edge (dy > 0) and dx < 0.

3. Exiting edge (dy < 0) and dx < 0.

4. Exiting edge (dy < 0) and dx > 0.
1. Entering edge (dy > 0) and dx > 0.

\[
\begin{array}{cccccccc}
\text{xLow} & \text{yLow} & \text{xHigh} & \text{yHigh} & \text{dx} & \text{dy} & \text{F} & \text{IntX} & \text{x} \\
\end{array}
\]

Initialize \( \text{IntX} = x = \text{xLow} \);
\[
\text{F} = \text{dy}.
\]

Update1(edge e)
{
   \[
   \text{F} = \text{F} + \text{dx};
   \]
   while (\( \text{F} > \text{dy} \)) {
      \[
      \text{F} -= \text{dy};
      \]
      \[
      \text{IntX}++;
      \]
   }
   \[
   x = \text{Intx};
   \]
\[
\begin{array}{c}
\text{dx} = 16 \quad \text{dy} = 6 \\
\text{F} = 6 \quad \text{IntX} = 0
\end{array}
\]
<table>
<thead>
<tr>
<th>xLow</th>
<th>yLow</th>
<th>xHigh</th>
<th>yHigh</th>
<th>dx</th>
<th>dy</th>
<th>F</th>
<th>IntX</th>
<th>x</th>
</tr>
</thead>
</table>

Initialize \( \text{IntX} = x = x_{\text{Low}}; \)
\[ F = dy; \]

4. Exiting edge \((dy < 0)\) and \(dx > 0\).

**Update4** (edge \(e\))
```
{ 
  F = F - dx;
  while (F < dy) {
    F -= dy;
    IntX--;
  }
  if (F == dy)
    x = Intx - 1;
  else x = IntX;
}
```

\(dx = 16\) \(dy = -6\)
\[ F = -6\] \(\text{IntX} = 16\)