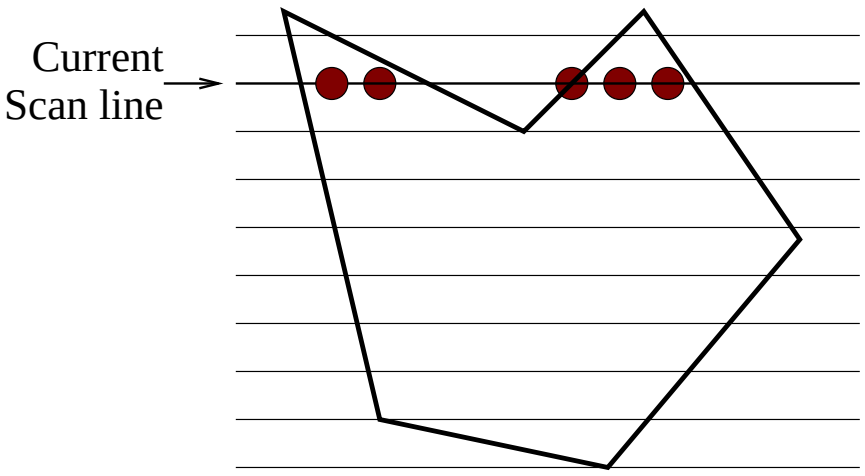
Rasterizing polygons

The filling of polygons is usually broken into two parts:

- Determining what pixels are interior to the poloygon, and therefore are to be filled
- Actually filling the polygon (accounting for shading, texturing, transparency, etc.)

We will consider here how to determine which pixels to fill.

We will do this by taking successive horizontal scan lines that intersect the primitive, and filling in the spans of pixels that are interior to the primitive. (This is similar to the way pixels are displayed on a CRT.)



Each span will be filled by finding the span extrema — the start and end points of the span. By specifying the span extrema,

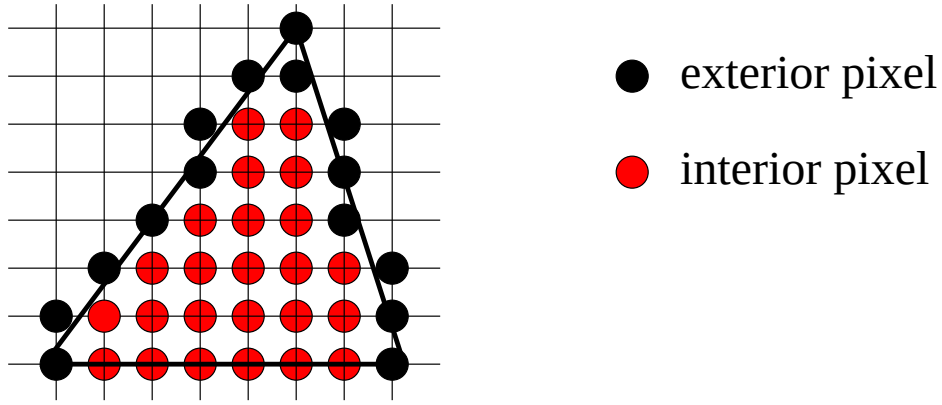
the span can be filled quite quickly. For example, if writing to a bilevel display, accessed by 8-bit words, a span of 5 pixels could be filled by writing the single word `11111000` to the frame buffer (this assumes that spans and memory words have been aligned). Writing the 5 pixels separately would be much slower. (We will not consider the details of this further — suffice to say that filling individual pixels is slow.)

Span algorithms exploit spatial coherence — the property that adjacent pixels do not change (unless at a polygon boundary.)

We will first consider the filling of simple, convex polygons like triangles. Here, a simple algorithm could be:

1. Draw the polygon boundaries as lines and label these pixels as boundary points
2. For each scan line, fill all pixels between the boundary points

Here is an example of the application of this simple agorithm:

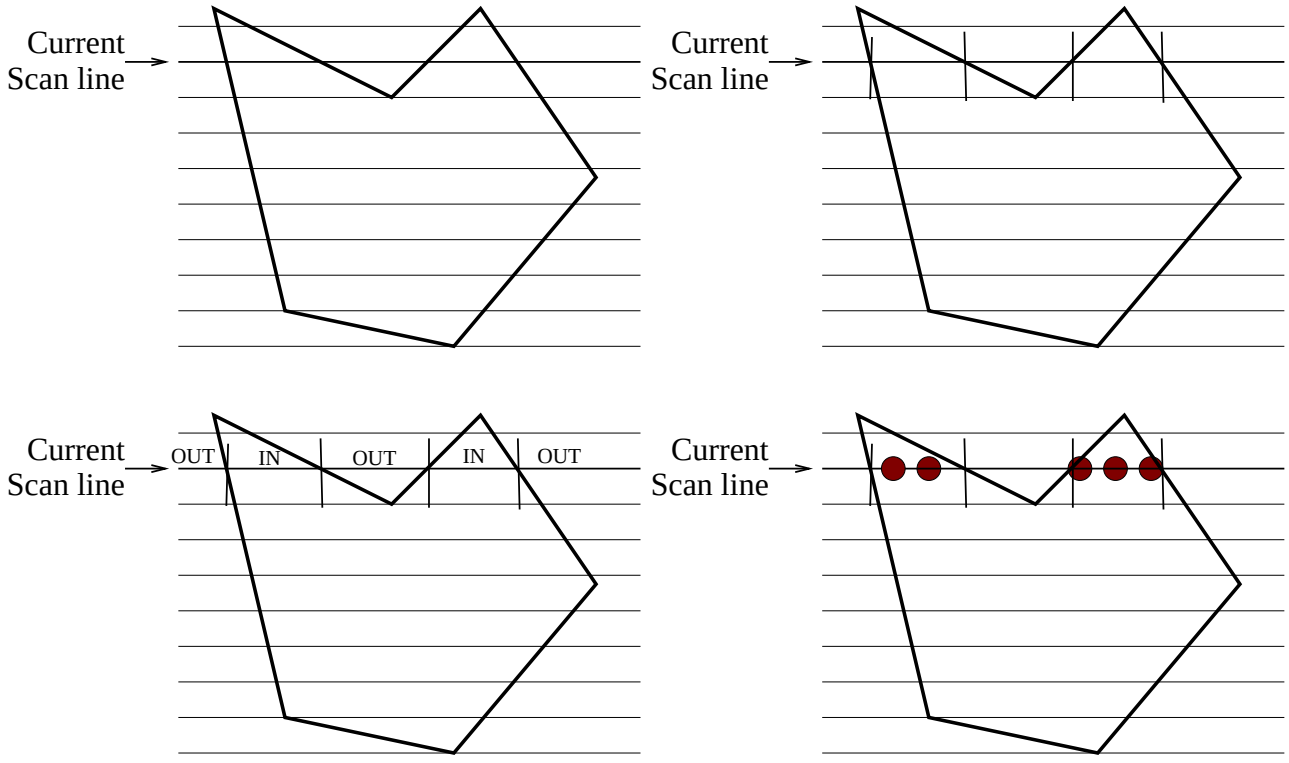


Note that there is a problem here; for a polygon, we want to fill interior pixels only. Otherwise we may get overlap artefacts for adjacent polygons. Therefore we cannot just "draw the boundary points" in a line. When a line is rasterized, some pixels will fall on either side of the "true" line.

We require a method of selecting boundary points that are strictly interior to the polygon.

We will pose a method that works for non-convex polygons, as well as convex polygons.

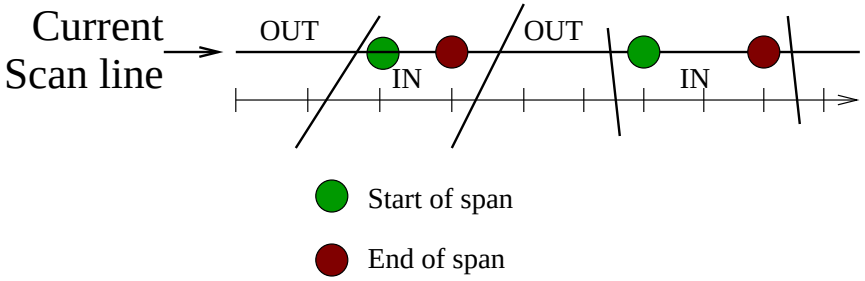
To extend to non-convex polygons, we need to keep an ordered list of intersections. A left-to-right traversal of these intersections (**not** the pixels) can then be used to determine whether each pixel is interior or exterior. A parity check is used for this purpose. First, the parity is set to even (meaning we are outside). Then for each intersection we flip the parity. We only fill spans whose parity is odd (meaning we are inside).



Again, there are some unanswered questions:

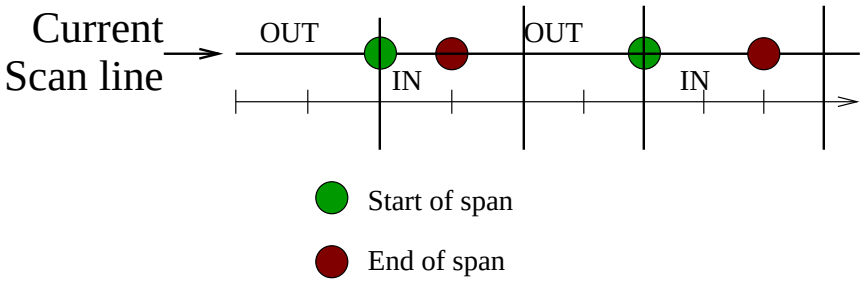
What do we do if the intersection point does not fall on an integer?

If the parity bit indicates that we are outside, we round up to set the start pixel of the next span. If we are inside, we round down to set the end of the current span.



What do we do if the intersection point does fall on an integer?

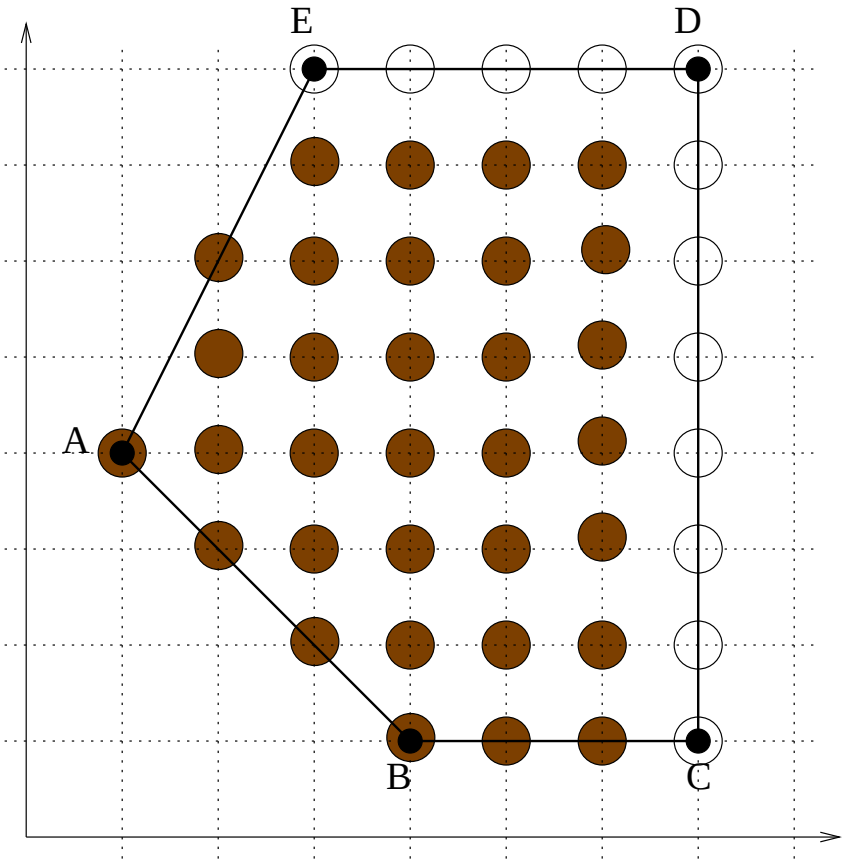
If the intersection point has integer coordinates, and we are exterior, then we define that integer value as the start pixel of the next span. If we are interior, then we define the end pixel of our span as the intersection's value minus one.



Thus, the left edge is considered part of the polygon, but the right edge is not.

What happens at shared vertices?

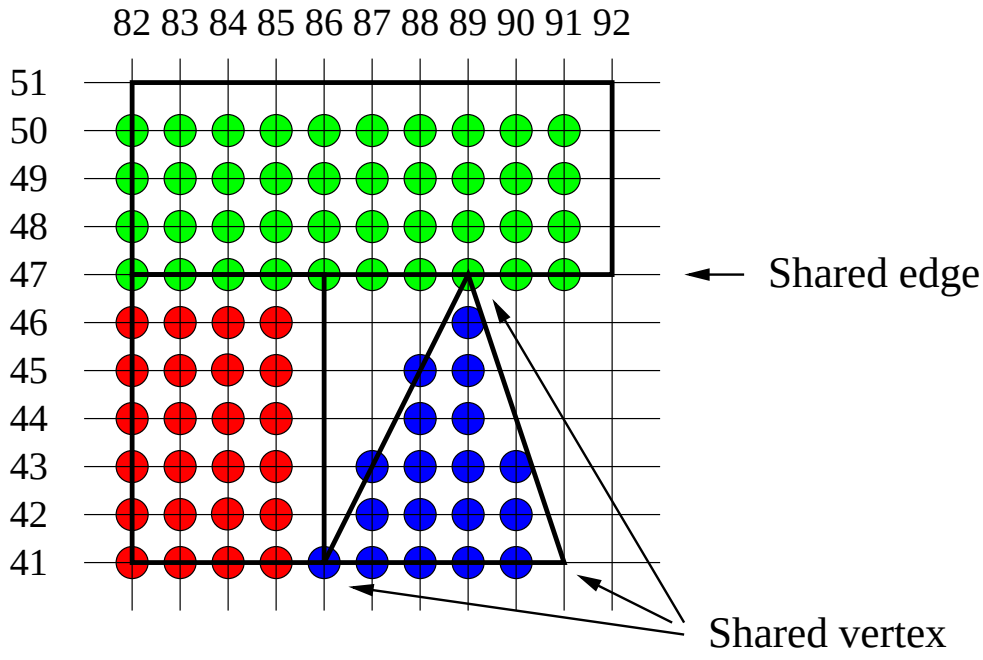
Each edge will have a y_{min} vertex (lower $y-$coordinate) and a y_{max} vertex (higher $y-$coordinate). Horizontal edges have neither. For shared vertices, the y_{min} vertex of an edge is counted (i.e. flips the parity bit), but not the y_{max} vertex.



Thus, if the shared vertex has at least one edge with a y_{min} vertex then it will start a span. (Example: vertex A) This leads to the fact that horizontal edges on the bottom of the polygon are drawn. (Example: line BC)

If the shared vertex has no edges with a y_{min} vertex then it will **not** start a span. This leads to the fact that horizontal edges on the top of the polygon are not drawn. (Example: line ED)

Another Example:



Note the shared edge between the red (lower) rectangle, and the green (upper) rectangle. It is colored green.

The lower shared vertex of the triangle is on its left edge, so it is part of the triangle (blue), not part of the (red) rectangle on the right edge.

The upper shared vertex of the triangle is the top, so it is part of the upper rectangle (green).

Scan-line intersections

So far, we have not considered the problem of computing the intersection of the scan line with the primitives — there are many scan lines, so this computation should be done efficiently. (Some extremely efficient algorithms are available for special classes of figures; rectangles, for example.)

This is where spatial coherence is useful — it is likely that most edges intersecting scan line i also intersect line $i + 1$. If we know the slope of the edge, we can use a method similar to the line rasterizing algorithm to determine the intersection with the next scan line.

~~Consider lines with $|m| \geq 1$.~~ *Consider non-horizontal edges.* Beginning at the lower vertex, we fill (x_0, y_0). We then increment y by 1 and determine x as follows:

$$x_{k+1} = x_k + 1/m$$

If we know the intersection of this edge with the previous scan line (x_k) then we can calculate the intersection with the next scan line (x_{k+1}). Of course, the calculated intersection may have to be rounded according to the rules given above. (A fast integer algorithm is presented in the text).

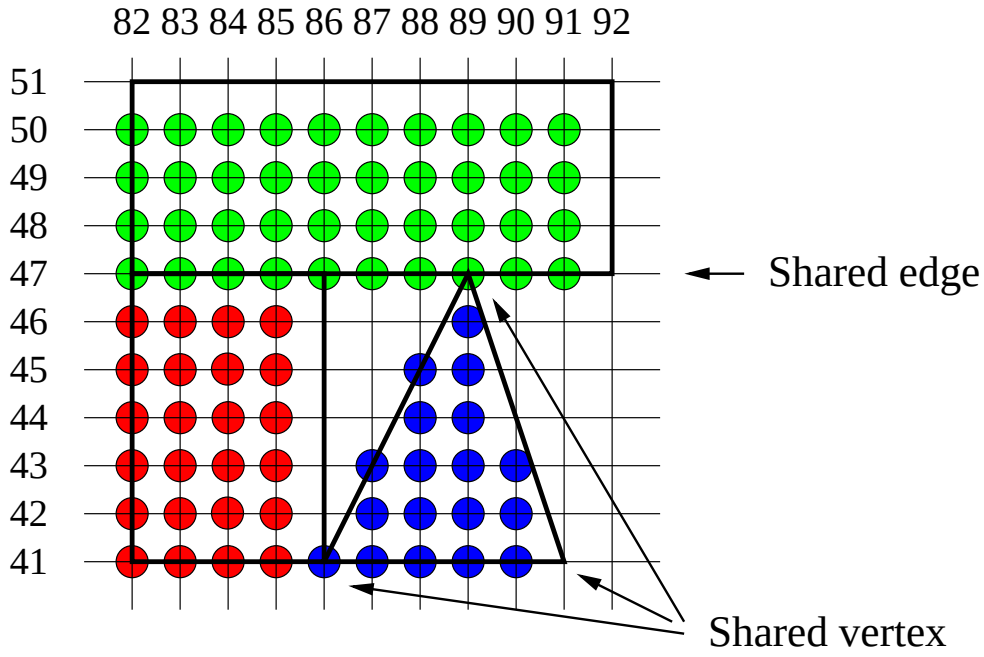
We can now develop a scan-line algorithm which takes advantage of this edge coherence. We need to keep track of the sets of edges that have (or will have) intersections with the present and upcoming scan lines.

We will create two tables holding information about polygon edges.

First, we create a global edge table (ET) containing all edges sorted by their y_{min} coordinate. There may be several edges with the same y_{min}, and those are sorted by their x_{min} coordinates (a bucket sort, where each bucket contains the values sorted by their x_{min} coordinate.)

The table index is y_{min} and the entries are the maximum y values y_{max} for each edge, the x value (initially set to x_{min}), and the increment for the x value $(1/m)$.

We create a second data table called the active edge table (AET) to which we will add the edges which will intersect with the scan line, and delete edges which will not.



Example ET with entries in the form $(y_{max}, x_{min}, 1/m)$:

41 $(47, 82, 0) \rightarrow (47, 86, 0) \rightarrow (47, 86, 1/2) \rightarrow (47, 91, -1/3)$

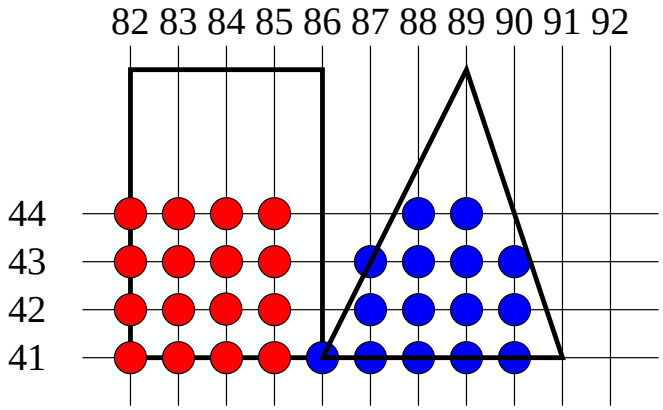
47 $(51, 82, 0) \rightarrow (51, 92, 0)$

(The rectangle and triangle begin on scan line 41, and have 2 edges each. The upper rectangle begins on line 47, and has 2 edges. Horizontal edges are not included.)

Once the ET has been formed, the following processing steps for the scan-line algorithm are completed:

1. Set y to the smallest y coordinate that has an entry in the ET, that is, y for the first nonempty bucket.
2. Initialize the AET to be empty.
3. Repeat until the AET and ET are empty;
   (a) Move from ET bucket y to the AET edges whose $y_{min} = y$ (add entering edges).
   (b) Remove from the AET those entries for which $y = y_{max}$ (edges not involved in the next scan line), then sort the AET on x.
   (c) Fill in desired pixel values on scan line y by using pairs of x coordinates from the AET (suitably rounded).
   (d) Increment y by 1 (to the coordinate of the next scan line).
   (e) For each nonvertical edge remaining in the AET, update x for the new y.

An example follows on the next page...



AET for y = 41:

$(47, \mathbf{82}, 0) \rightarrow (47, \mathbf{86}, 0) \rightarrow (47, \mathbf{86}, 1/2) \rightarrow (47, \mathbf{91}, -1/3)$

Fill spans 82-85 and 86-90.

AET for y = 42:

$(47, \mathbf{82}, 0) \rightarrow (47, \mathbf{86}, 0) \rightarrow (47, \mathbf{86.5}, 1/2) \rightarrow (47, \mathbf{90.666}, -1/3)$

Fill spans 82-85 and 87-90.

AET for y = 43:

$(47, \mathbf{82}, 0) \rightarrow (47, \mathbf{86}, 0) \rightarrow (47, \mathbf{87.0}, 1/2) \rightarrow (47, \mathbf{90.333}, -1/3)$

Fill spans 82-85 and 87-90.

AET for y = 44:

$(47, \mathbf{82}, 0) \rightarrow (47, \mathbf{86}, 0) \rightarrow (47, \mathbf{87.5}, 1/2) \rightarrow (47, \mathbf{90.0}, -1/3)$

Fill spans 82-85 and 88-89.

...etc...