

On-Line Computer Graphics Notes

RASTERIZING POLYGONS IN IMAGE SPACE

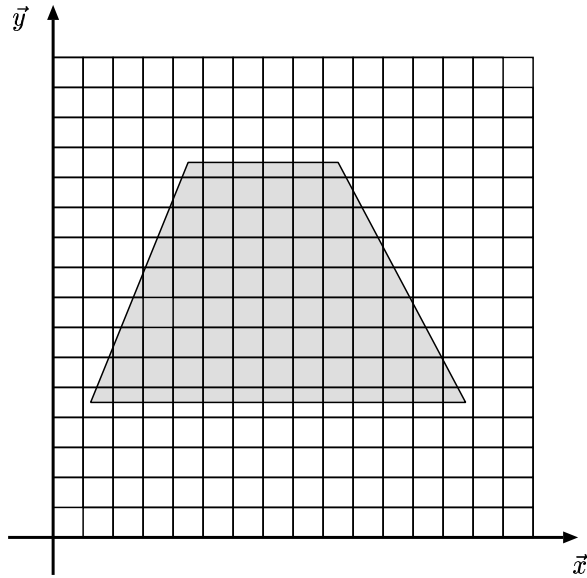
Kenneth I. Joy
Visualization and Graphics Research Group
Department of Computer Science
University of California, Davis

A fundamental process in computer graphics and visualization is the process of *scan conversion* or *rasterization*. Given a polygon in image space, this process determines the pixels that intersect the polygon. This process is utilized in visible-surface algorithms, incremental-shading techniques, polygon-fill algorithms, ray-tracing-acceleration algorithms, and a number of other tasks that are critical to the understanding of the computer graphics field.

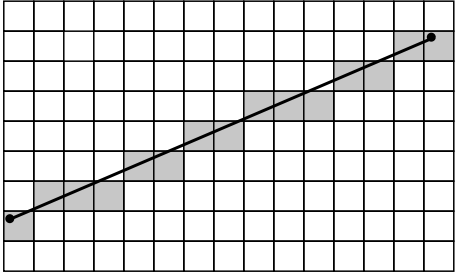
The conventional way to calculate the pixels that intersect a polygon is to utilize what is commonly called a “digital differential analyzer” or DDA. The term DDA refers to an antiquated mechanical device that solves differential equations by numerical methods. The term has stuck since the methods of approximating values by simultaneously incrementing by small steps is exactly what these algorithms do. The basic DDA algorithm utilized for rasterization is Bresenham’s Algorithm.

1 Rasterization of Trapezoids in Device Space I

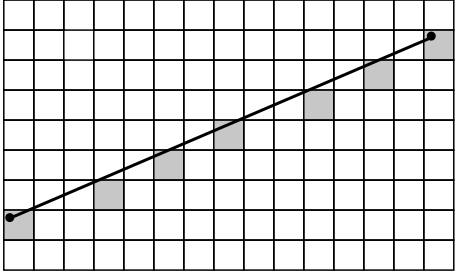
Rasterization is the process of finding the pixels in device space that correspond to a polygon in screen space. In this section we utilize Bresenham’s algorithm to assist in the scan conversion of specialized polygonal shapes – trapezoids in device space with their top and bottom edges parallel – each having the form $y = \text{constant}$. Any polygon in device space can be converted to a set of trapezoid of this form.



The idea for utilizing Bresenham's Algorithm in rasterization is to observe that this algorithm produces exactly one pixel per element of the driving axis. The figure below illustrates a line drawn with Bresenham's algorithm. In this case, we note that the x axis is the driving axis, and that any column of pixels that intersects the line, intersects only one illuminated pixel.

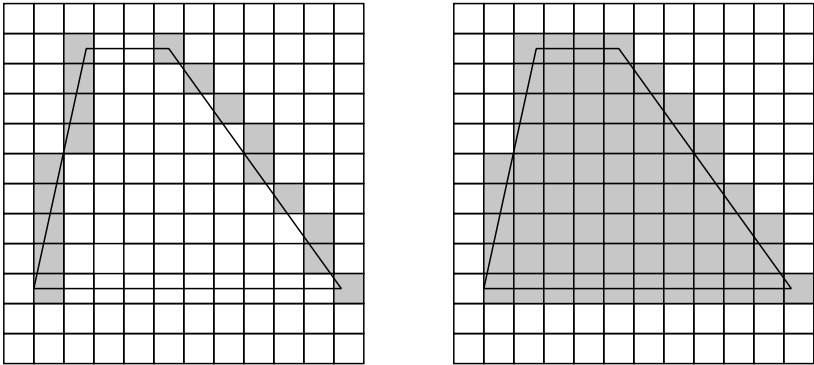


The following illustration shows the same line, but drawn as if we utilized the y axis as the driving axis.



We note now, that any row of pixels that crosses the line intersects only one illuminated pixel. This is the main idea behind adapting Bresenham's algorithm to be used in rasterization. We first need to adapt this algorithm so that the driving axis can always be the y -axis, and then develop a procedure to use the algorithm on each non-horizontal edge of the trapezoid. In this way, we will get exactly two illuminated pixels per row and can complete our rasterization process by just selecting all the pixels in a row bounded by the two pixels produced by our algorithm.

The following figure illustrates this process. The left-hand picture shows the pixels that have been illuminated by the two Bresenham's algorithms. The right-hand picture shows the final result after filling the rows between the two illuminated pixels.



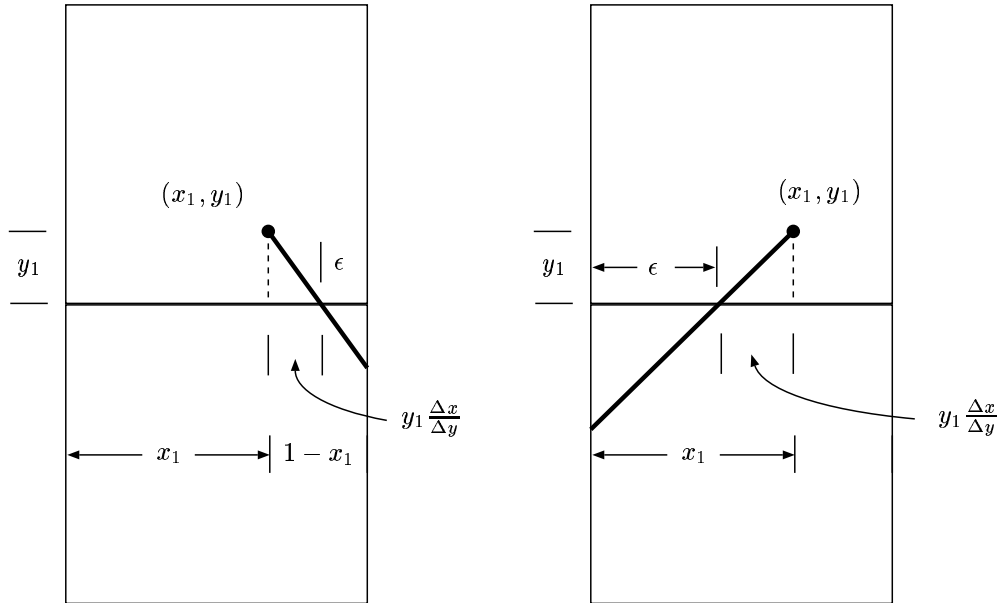
Now, how do we modify Bresenham's algorithm to always use the y -axis as the driving axis? Normally, within the main loop of the algorithm, the coordinate corresponding to the DA is incremented by one unit and the coordinate corresponding to the other axis need only be incremented occasionally. In this case, the coordinate on the DA can still be incremented by one unit, however the coordinate on the other axis may be incremented several times. This is actually a simple change to the algorithm, and can be implemented by replacing the statement

if ($\bar{\epsilon} \geq 0$)

with

while ($\bar{\epsilon} \geq 0$)

The algorithm now appears as follows¹ We also note we have changed the algorithm to reflect that the y axis is the driving axis. We note that $m = \frac{\Delta x}{\Delta y}$, and that according to the following picture



we must consider two possible initial values for ϵ , one if $\Delta x > 0$ (the left-hand illustration) and one if $\Delta x < 0$ (the right-hand illustration). Referring to the illustration, we have either

$$\epsilon = -(1 - x_1 - y_1 \frac{\Delta x}{\Delta y})$$

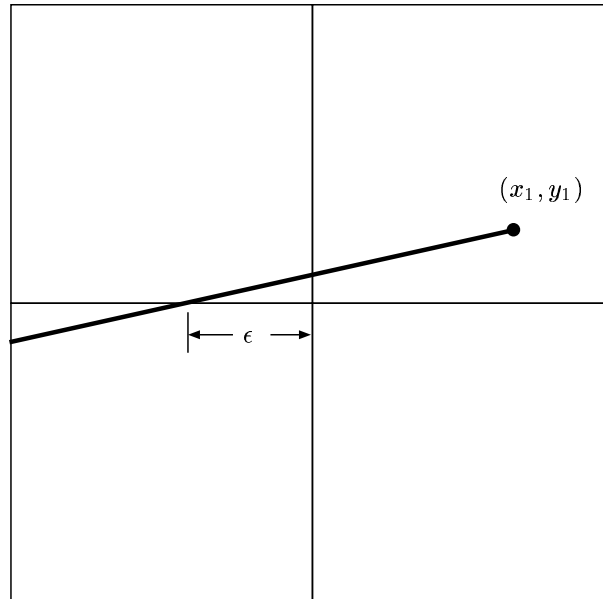
in the first case, or

$$\epsilon = -(x_1 - y_1 \frac{\Delta x}{\Delta y})$$

in the second. In general x_1 must be replaced by $x_1 - \lfloor x_1 \rfloor$ and y_1 by $y_1 - \lfloor y_1 \rfloor$ as the figure is drawn as if the lower-left-hand corner of the pixel is $(0, 0)$.

We also need to recognize that ϵ may be greater than zero immediately, as in the following figure. Therefore, we must move the “illuminate” step in our algorithm below the “while” statement. This will insure that we are at the correct boundary pixel for the trapezoid.

¹We note that we are using the non-integer form of the algorithm, which is clearer in its presentation. The conversion to the integer algorithm is straightforward.



The algorithm now appears as follows:

Bresenham's Algorithm with y as the Driving Axis

The points (x_1, y_1) and (x_2, y_2) are assumed not equal
 ϵ is assumed to be real.

Let $\Delta x = x_2 - x_1$

Let $\Delta y = y_2 - y_1$

Let $m = |\frac{\Delta x}{\Delta y}|$

Let $i = \lfloor x_1 \rfloor$

Let $j_1 = \lfloor y_1 \rfloor$

Let $j_2 = \lfloor y_2 \rfloor$

if $\Delta x > 0$

$i_{inc} = 1$

$\epsilon = -(1 - (x_1 - i) - (y_1 - j_1) \frac{\Delta x}{\Delta y})$

else

$i_{inc} = -1$

$\epsilon = -((x_1 - i) - (y_1 - j_1) \frac{\Delta x}{\Delta y})$

end if

for $j = j_1$ to j_2

while $(\epsilon \geq 0)$

$i += 1$

$\epsilon -= 1.0$

end while

illuminate (i, j)

$j += 1$

$\epsilon += m$

next j

finish

This algorithm will continue to increment the x value (and decrement ϵ) as long as ϵ is greater than zero. It illuminates only one pixel per row on the line.

We are assuming that $\Delta y \neq 0$ which is valid since we are only considering the non-horizontal edges of the trapezoids that we want to rasterize. It should also be mentioned that the above algorithm is written for the case where m is positive. If $m = 0$, then $\epsilon = -(1 - (x_1 - i))$ and is never incremented. In this case, we have a vertical line, and the x value will never need to be incremented. If $m < 0$, then we must decrement i in the algorithm, and add $|m|$ to ϵ .

To utilize the above algorithm to produce a rasterization of a trapezoid, we do the following steps.

- Run the modified Bresenham algorithm on the left edge of the trapezoid, keeping track of the “illuminated” pixels.
- Run the modified Bresenham algorithm on the right edge of the trapezoid, keeping track of the “illuminated” pixels.
- Illuminate the interior pixels from left to right on each row that has been illuminated.

Since we consider two possible initial values for ϵ , we can guarantee that this algorithm produces a rasterization where the union of the set of pixels produced by the algorithm contains the trapezoid.

It is not too difficult to see how to lump these three steps into one algorithm which computes the two endpoints on each row and fills between them. This is the topic of the next section

2 Rasterization of Trapezoids in Device Space II

We have shown in the previous section that we can modify Bresenham's algorithm to produce a rasterization of a trapezoid. In short, two Bresenham algorithms were run, each on one non-horizontal side of the trapezoid, and then the rows were filled by illuminating pixels between those generated by the two algorithms. In this section we modify the above process to produce a complete row of pixels with each iteration of one algorithm.

We construct this algorithm in a more general form, defining an *endpoint node* which contains the information on each endpoint of a row of pixels. This method will be useful, not only in rasterization, but as controlling mechanisms for visible-surface algorithms.

We create an endpoint node that retains the information that our algorithm needs to produce the rasterization of a line. The node will contain (at least) the following information.

Endpoint Node
j
i
i_{inc}
m
ϵ

We will assume that the y axis is the driving axis, and that the algorithm will proceed from the top to the bottom of the trapezoid. We will define two procedures: *initialize* and *update*, which will allow us to easily describe the total algorithm.

If we are given two points (x_1, y_1) and (x_2, y_2) in device space, with $y_1 > y_2$, the *initialize* procedure sets up the endpoint node as follows:

Initialize Procedure

Parameters – a line segment in device space specified by
 (x_1, y_1) and (x_2, y_2)

Returns – an **Endpoint** node

Let $j = \lfloor y_1 \rfloor$

Let $i = \lfloor x_1 \rfloor$

Let $\Delta x = x_2 - x_1$

Let $\Delta y = y_1 - y_2$

Let $m = \left| \frac{\Delta x}{\Delta y} \right|$

if $\Delta x > 0$

$i_{inc} = 1$

$\epsilon = -(1 - (x_1 - i) - (y_1 - j) \frac{\Delta x}{\Delta y})$

else

$i_{inc} = -1$

$\epsilon = -((x_1 - i) - (y_1 - j) \frac{\Delta x}{\Delta y})$

end if

while $(\epsilon \geq 0)$

$i += i_{inc}$

$\epsilon -= 1.0$

end while

The last four lines of the algorithm corrects the endpoint if ϵ is initially greater than zero. This step is necessary to insure that the complete trapezoid is contained in the union of the set of pixels that are produced by the rasterization algorithm.

The *update* procedure is utilized to make the modifications to the endpoint node as the algorithm proceeds from row to row. Given a node with j, i, i_{inc}, m and ϵ , the update procedure executes as follows:

Update Procedure

Parameters – an endpoint node
specifying j, i, i_{inc}, m and ϵ

Let $j - = 1$

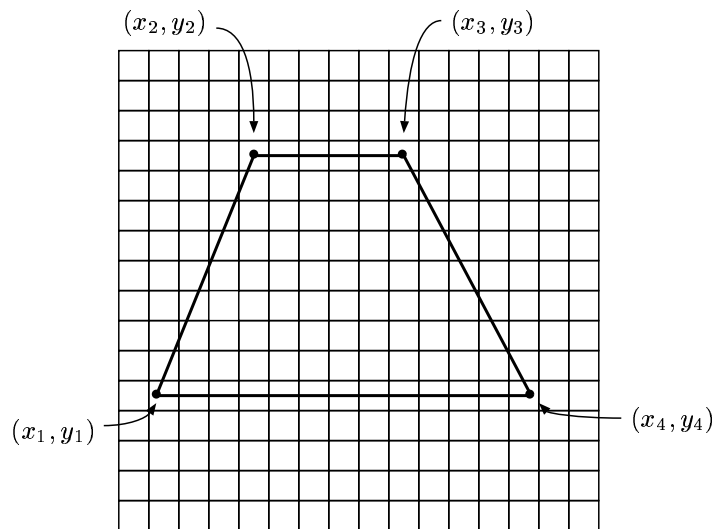
if $(\epsilon \geq 0)$

Let $i += i_{inc}$

Let $\epsilon - = 1.0$

Let $\epsilon + = m$

Now, given a trapezoid with the four corner points (x_1, y_1) , (x_2, y_2) , (x_3, y_3) and (x_4, y_4) , (as in the following figure),



We note that $y_2 = y_3$ and $y_1 = y_4$, and the algorithm proceeds as follows:

Rasterization Algorithm for Trapezoids in Device Space

Endpoint $e_1 = \text{Initialize} ((x_2, y_2), (x_1, y_1))$

Endpoint $e_2 = \text{Initialize} ((x_3, y_3), (x_4, y_4))$

for $j = y_2$ to y_1

for $i = e_1.i$ through $e_2.i$

illuminate (i, j)

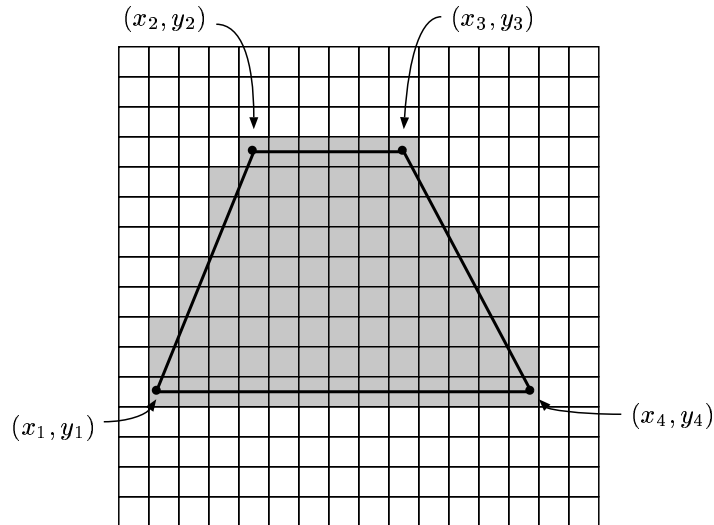
Update (e_1)

Update (e_2)

next j

finish

producing the following rasterization of the trapezoid.



Thus, by defining the endpoint nodes, we can specify a straightforward rasterization procedure for trapezoids.

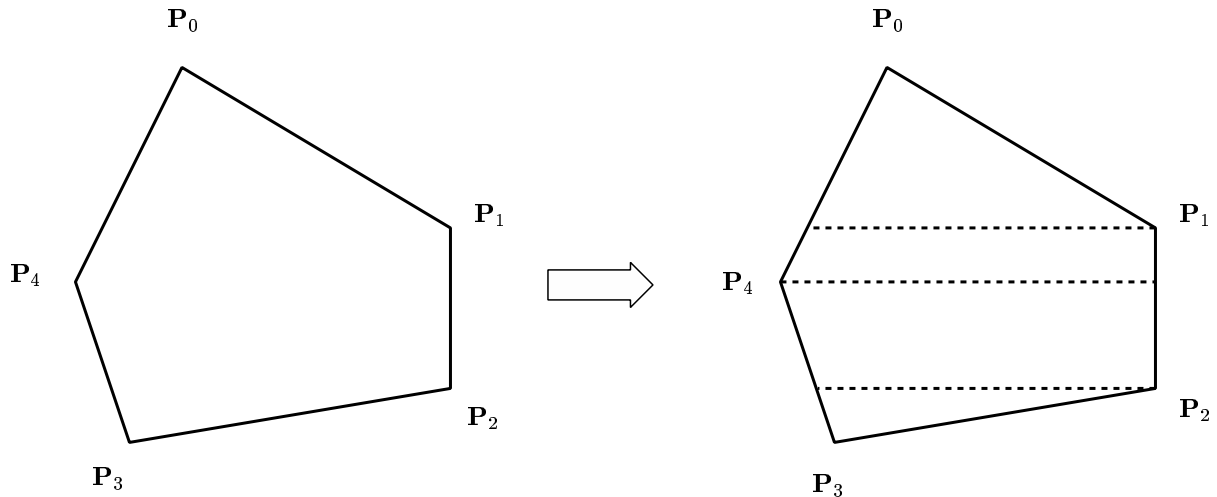
This concept of a “endpoint node” is important for many uses of rasterization. In particular, a number of visible-surface algorithms utilize this concept as a fundamental part of the algorithm.

3 Rasterization of Polygons in Device Space

Given a convex polygon in device space, it can be easily converted to a set of trapezoids by the following procedure

- Let y_{max} and y_{min} be the maximum and minimum y values that are taken on by the polygon (these extrema must exist at some of the vertices of the polygon, so they are easy to find).
- For those vertices that do not have a y value equal to y_{min} or y_{max} , draw horizontal lines across the polygon to create trapezoids.

The following figure illustrates this procedure. The left picture is of a polygon in device space and the right picture is the trapezoidal decomposition of the polygon



4 Rasterization of Polygons in Extended Device Space

In extended device space polygons are three dimensional – they have depth. We utilize this depth information in our visible-surface algorithms. Rasterization in this space consists of not only finding the pixels that best represent the polygon, but also finding a z value that represents the “depth” of the pixel.

Since polygons are planar, any polygon can be split into a set of trapezoids, whose top and bottom edges are parallel and have constant y value (In extended device space, this implies that these edges must lie in a plane with equation $y = \text{constant}$.) We have already developed algorithms for these trapezoids in device space – to expand them to work in extended device space is straightforward, as we only need consider increments for the z value.

Consider a polygon in extended device space and let \vec{n} be its normal. We know that for any two points \mathbf{Q}_1 and \mathbf{Q}_2 in the plane of the polygon, we have

$$\vec{n} \cdot (\mathbf{Q}_2 - \mathbf{Q}_1) = 0$$

since $\mathbf{Q}_2 - \mathbf{Q}_1$ is a vector in the plane.

Now, consider the process of rasterization in extended device space. We will still increment the x and y coordinates by one in our algorithm, however, we now have a z coordinate, and when incrementing this coordinate, we are constrained that the resulting point must still lie in the plane of the polygon. That is, if

(x, y, z) is our initial point, then both $(x + 1, y, z + z_{xinc})$ and $(x, y + 1, z + z_{yinc})$ must be in the plane of the polygon.

So let $\vec{n} = \langle x_n, y_n, z_n \rangle$, and consider the points on the polygon (x, y, z) and $(x + 1, y, z + z_{xinc})$. By our above discussion, $(x + 1, y, z + z_{xinc}) - (x, y, z)$ is a vector in the plane of the polygon and therefore

$$\begin{aligned} 0 &= \langle x_n, y_n, z_n \rangle \cdot ((x + 1, y, z + z_{xinc}) - (x, y, z)) \\ &= \langle x_n, y_n, z_n \rangle \cdot \langle 1, 0, z_{xinc} \rangle \\ &= x_n + z_n z_{xinc} \end{aligned}$$

which implies that

$$z_{xinc} = -\frac{x_n}{z_n}$$

Alternatively, if we consider the two points (x, y, z) and $(x, y + 1, z + z_{yinc})$. Our above discussion implies that $(x, y + 1, z + z_{yinc}) - (x, y, z)$ is a vector in the plane, and

$$\begin{aligned} 0 &= \langle x_n, y_n, z_n \rangle \cdot ((x, y + 1, z + z_{yinc}) - (x, y, z)) \\ &= \langle x_n, y_n, z_n \rangle \cdot \langle 0, 1, z_{yinc} \rangle \\ &= y_n + z_n z_{yinc} \end{aligned}$$

which implies that

$$z_{yinc} = -\frac{y_n}{z_n}$$

Thus, the z increments can be determined directly from the vector that is normal to the plane of the polygon.

We note that if $z_n = 0$, we have a problem – as we have constructed a potential divide-by-zero situation. However, this problem should not surface, since if the z component of the normal was zero, then the polygon would be edge-on to device space and would not be seen.

We now extend our *endpoint node* to include the z information of extended device space. The two increments z_{xinc} and z_{yinc} are also included in the node, even though they are fixed for the entire polygon and thus they could be considered global. However, as we encounter cases where many polygons may be

rasterized in parallel, it is common to put this information into the node. Thus, we obtain

Endpoint Node
j
i
i_{inc}
z
z_{xinc}
z_{yinc}
m
ϵ

We modify the Initialize procedure by integrating the initial calculation of, and the modification of the z coordinate.

Initialize Procedure

Parameters – a line segment in device space specified by (x_1, y_1, z_1) and (x_2, y_2, z_2) , and a normal vector $\langle x_n, y_n, z_n \rangle$

Returns – an **Endpoint** node

Let $j = \lfloor y_1 \rfloor$

Let $i = \lfloor x_1 \rfloor$

Let $z = z_1$

Let $\Delta x = x_2 - x_1$

Let $\Delta y = y_1 - y_2$

Let $m = \left| \frac{\Delta x}{\Delta y} \right|$

Let $z_{xinc} = -\frac{x_n}{z_n}$

Let $z_{yinc} = -\frac{y_n}{z_n}$

if $\Delta x > 0$

$i_{inc} = 1$

$\epsilon = -(1 - (x_1 - i) - (y_1 - j) \frac{\Delta x}{\Delta y})$

else

```

iinc = -1
zxinc = -zxinc
ε = -((x1 - i) - (y1 - j)  $\frac{\Delta x}{\Delta y}$ )
end if

while (ε ≥ 0)
  i + = iinc
  z + = zxinc
  ε - = 1.0
end while

```

The update procedure can be modified in a similar way, in that we must modify the *z* coordinate by incrementing it by either *z*_{xinc} or *z*_{yinc}. We obtain

Update Procedure

Parameters – an endpoint node

specifying *j*, *i*, *z*, *z*_{xinc}, *z*_{yinc}, *i*_{inc}, *m* and *ε*

Let *j* - = 1

Let *z* - = *z*_{yinc}

if (*ε* ≥ 0)

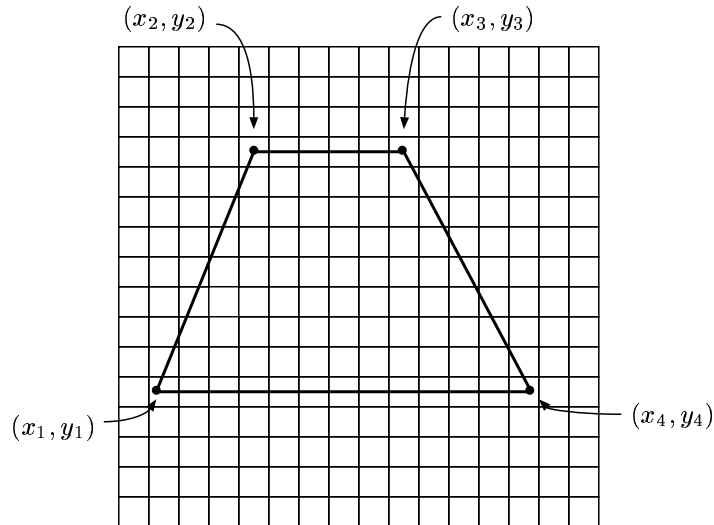
Let *i* + = *i*_{inc}

Let *z* + = *z*_{xinc}

Let *ε* - = 1.0

Let *ε* + = *m*

Combining the Initialize and Update procedures, we obtain our algorithm. Given the following polygon in extended device space.



The algorithm appears as follows:

Rasterization Algorithm for Trapezoids in Extended Device Space

```

Let  $\vec{v}_1 = (x_2, y_2, z_2) - (x_1, y_1, z_1)$ 
Let  $\vec{v}_2 = (x_4, y_4, z_4) - (x_1, y_1, z_1)$ 
Let  $\vec{n} = \vec{v}_1 \times \vec{v}_2$ 

Endpoint  $e_1 = \text{Initialize} ( (x_2, y_2, z_2), (x_1, y_1, z_1), \vec{n} )$ 
Endpoint  $e_2 = \text{Initialize} ( (x_3, y_3, z_3), (x_4, y_4, z_4), \vec{n} )$ 

for  $j = y_2$  to  $y_1$ 
  for  $i = e_1.i$  through  $e_2.i$ 
    illuminate  $(i, j)$ 

  Update  $(e_1)$ 
  Update  $(e_2)$ 
  next  $j$ 

finish

```

We will utilize this procedure repeatedly in our computer graphics algorithms. In addition to the z coordinate, we frequently utilize quantities relating to color, texture, and parameterization as information in our endpoint nodes and increment them as well. These quantities are included in the algorithm in a manner similar to the z coordinate above.

**All contents copyright (c) 1996, 1997, 1998, 1999
Computer Science Department, University of California, Davis
All rights reserved.**