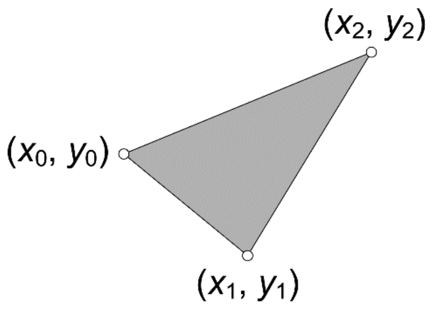
ITCS 481 2/2020 Episode 1: The Triangle Awakens

Do not submit source code from other students or web sites.

Implement triangle scan conversion to a simulated frame buffer and smoothly shade triangles from three different vertex colors. Vertices will be specified by mouseclicks.

Code on *mycourses* will show how a frame buffer should be simulated and how mouse button events are received. Extend this to store successive mouse click coordinates and to call your scan conversion routine for every three vertices specified. Do *not* use OpenGL commands for triangle drawing. Draw only by setting color values in the simulated frame buffer.

Your implementation *must scan by columns (instead of rows)* and use the following methods:



Given three vertices (x_0, y_0), (x_1, y_1), (x_2, y_2),
such that $x_0 \leq x_1 \leq x_2$, and
letting $L_{i,j}$ denote the edge from (x_i, y_i) to (x_j, y_j),

```
For each column from $x_0$ to $x_1$-1:
        Compute the two y-coords where $L_{0,1}$ and $L_{0,2}$
        intersect this column, and color pixels in this range

For each column from $x_1$ to $x_2$:
        Compute the two y-coords where $L_{1,2}$ and $L_{0,2}$
        intersect this column, and color pixels in this range
```

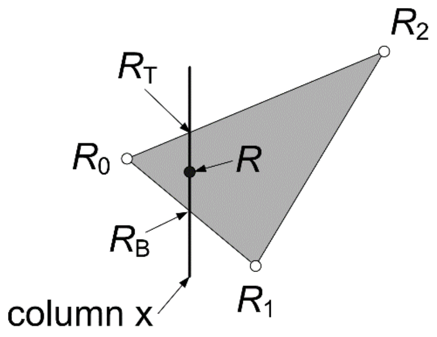
Some details are omitted by the above high-level algorithm:

- Make it work for any 3 vertices, regardless of order. To guarantee $x_0 \leq x_1 \leq x_2$, the three vertices can be passed to scan conversion in corrected order, or sorted (it's usually best to not waste time with swaps).
- Calculation of intersection y-coords should be done efficiently by using an incremental approach like DDA. For example: at column x_0, $L_{0,1}$ intersects the column with y-coordinate y_0, so y_0 is the initial value for $L_{0,1}$ intersection. Then, each time the process moves right one column, a new y-coordinate of intersection is found by adding slope to the previous value (slope is a floating point ratio and should be precomputed). $L_{0,2}$ and $L_{1,2}$ intersections can be handled by a similar method. At column x_1, stop tracking the $L_{0,1}$ intersection and initialize the $L_{1,2}$ intersection.
- Edge intersection coordinates are not generally integers. For a column, color the pixels from $\lceil y_B \rceil$ to $\lfloor y_T \rfloor$, where y_B and y_T are y-coordinates of the two intersections and ($y_B \leq y_T$).
- If $x_0 = x_1$ or $x_1 = x_2$, there is a vertical edge, so computing slope could lead to divide-by-zero. This is easily checked. When $x_0 = x_1$, the first "half" of the algorithm is skipped. When $x_1 = x_2$, the second half should just draw one line (column).
- In the diagram above and in the discussion below, $L_{0,1}$ and $L_{1,2}$ are below $L_{0,2}$. You must also handle cases with $L_{0,2}$ below $L_{0,1}$ and $L_{1,2}$.

CREDIT: The homework was written by Dr. Christoph W. Borst for his CMPS 415 course at the University of Louisiana Lafayette.

Smoothly shading the triangle:

Suppose each vertex (x_i, y_i) has a color value (R_i, G_i, B_i) of your choosing. The discussion below explains how to compute the red level R at a generated pixel. A similar process can be used to compute green and blue levels.



Suppose scan conversion has just generated the illustrated interior pixel, and call its coordinate (x, y). First interpolate red levels along bottom and top edges to compute levels R_B and R_T for the column at x. Then compute the final red level, R, by interpolating between R_B and R_T.

Do *not* use any direct interpolation formula. Instead, use increments for efficiency. For example, at column x_0, R_B is initialized to R_0. Then, each time scan conversion moves right one column towards x_1, R_B is incremented by adding $[(R_1\text{-}R_0)/(x_1\text{-}x_0)]$ to its previous value (the increment is the rate of change, i.e., amount of change in red per column). A similar process is used for R_T.

Then, as the pixels in the column are filled, R is first initialized and then incremented by $\Delta_R = [(R_T - R_B)/(y_T - y_B)]$ for each upward move. Recall that y_B and y_T are not necessarily integers. So, R is not initialized to R_B. Instead, it is initialized to $[R_B + (\lceil y_B \rceil - y_B) * \Delta_R]$ to account for the distance between (x, y_B) and the actual bottom pixel coordinate (x, $\lceil y_B \rceil$).