Brad Burkman's Notes for

CSCE 515 Principles of Computer Graphics

Dr. Christoph Borst, CBDI co-PI, OLVR 342, cbx99999@louisiana.edu

Fall 2018

Grader: Jason Woodworth, downtothelastpixel@gmail.com

Contents

1 Tuesday 21 August: Introduction

Ray Tracing

OpenGL 3.3

Couldn't find a textbook that did both the math and OpenGL 3.3 well.

Cross Product v/s Dot Product of Vectors

Start with how lines and triangles are rendered.

Acknowledge sources I used in homework and projects.

Exam will be hard.
Lagniappe - Come to VR lab to do a project.

2 Math Review: Dot Product and Cross Product

Example of Dot Product and Cross Product:
$\vec{u} = \langle 1, 2, 3 \rangle$
$\vec{v} = \langle 4, 5, 6 \rangle$

$\vec{u} \cdot \vec{v} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 4 + 10 + 18 = 32$

$$\vec{w} = \vec{u} \times \vec{v} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{vmatrix} = \begin{vmatrix} 2 & 3 \\ 5 & 6 \end{vmatrix} \vec{i} - \begin{vmatrix} 1 & 3 \\ 4 & 6 \end{vmatrix} \vec{j} + \begin{vmatrix} 1 & 2 \\ 4 & 5 \end{vmatrix} \vec{k} = -3\vec{i} + 6\vec{j} - 3\vec{k} = \langle -3, 6, -3 \rangle$$

Is the cross product commutative?

$$\vec{v} \times \vec{u} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{vmatrix} = \begin{vmatrix} 5 & 6 \\ 2 & 3 \end{vmatrix} \vec{i} - \begin{vmatrix} 4 & 6 \\ 1 & 3 \end{vmatrix} \vec{j} + \begin{vmatrix} 4 & 5 \\ 1 & 2 \end{vmatrix} \vec{k} = 3\vec{i} - 6\vec{j} + 3\vec{k} = \langle 3, -6, 3 \rangle = -\vec{w}$$

No, it's anti-commutative, but that's okay. The cross product gives a vector, $\vec{w}$, orthogonal to both $\vec{u}$ and $\vec{v}$, and a constant multiple of $\vec{w}$ is still orthogonal to both other vectors.

The cross product is also not associative, but satisfies the Jacobi identity.

$$a \times (b \times c) + b \times (c \times a) + c \times (a \times b) = 0 \ \forall a, b, c \in V$$

3 Math Review: Algebra

A **group** is a set S with an operation ("+") such that:

The set S is *closed* under the operation, meaning that if $a, b \in S$, then $a + b \in S$.
The operation is associative, meaning that if $a, b, c \in S$, then $a + (b + c) = (a + b) + c$.
The set S contains a unit element ("0"), such that $a + 0 = 0 + a = a \ \forall \ a \in S$.

For every element $a \in S$, there is an inverse element, $-a$, such that $a+(-a) = -a+a = 0$.
If the operation is commutative, meaning $a + b = b + a \ \forall \ a, b, \in S$, then S is called an **Abelian group.**

A **ring** is a set R with two operations, $+$ and $\cdot$, such that:

The set R is an Abelian group.

The set R is closed under multiplication.

Multiplication is associative.

The distributive laws hold:

$a \cdot (b + c) = a \cdot b + a \cdot c$

$(b + c) \cdot a = b \cdot a + c \cdot a$

A ring with a multiplicative identity ("1") is called a **ring with unit**.
If the ring has the property that if $a \cdot b = 0$ then $a = 0$ or $b = 0$, it is called a **domain**.
If multiplication is commutative, then the ring is called a **commutative ring**.
If a domain is commutative, then it is called an **integral domain**.

A **field** is a commutative ring with unit element ("1") such that every nonzero element has an inverse.

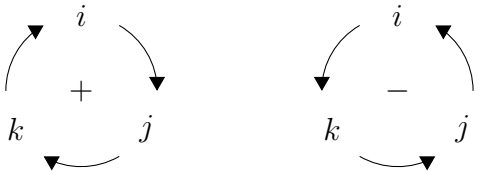
Quaternions are not a field because multiplication of quaternions is not commutative; however, every nonzero quaternion has a multiplicative inverse, so they are called a **division ring**.

4 Math Review: Quaternions

Complex numbers, $a+bi$ where a and b are real numbers and $i = \sqrt{-1}$, are two-dimensional numbers. Quaternions, $a+bi+cj+dk$, are four-dimensional numbers. Just as you can think of complex numbers being two-dimensional vectors having the basis vectors $1 = \langle 1, 0 \rangle$ and $i = \langle 0, 1 \rangle$, you can think of quaternions as four-dimensional vectors having the basis vectors $1 = \langle 1, 0, 0, 0 \rangle$, $i = \langle 0, 1, 0, 0 \rangle$, $j = \langle 0, 0, 1, 0 \rangle$, $k = \langle 0, 0, 0, 1 \rangle$,

While complex numbers have the basis elements 1 and i with, $i^2 = -1$, quaternions have this multiplication scheme for their basis elements. Note that $i^2 = j^2 = k^2 = ijk = -1$, but they are anti-commutative, with, for example, $ij = -ji$.

$\times$	1	i	j	k
1	1	i	j	k
i	i	-1	k	$-j$
j	j	$-k$	-1	i
k	k	j	$-i$	-1



The inverse of a quaternion is given by:

$$(a + bi + cj + dk)^{-1} = \frac{a - bi - cj - dk}{a^2 + b^2 + c^2 + d^2}$$

Vector Form of Quaternions
Think of $a + bi + cj + dk$ as the pair, $(a, \langle b, c, d\rangle)$, with a scalar part and a vector part.
Then quaternion addition is $(r_1, \vec{v}_1) + (r_2, \vec{v}_2) = (r_1 + r_2, \vec{v}_1 + \vec{v}_2)$.
Vector multiplication is $(r_1, \vec{v}_1)(r_2, \vec{v}_2) = (r_1 r_2 - \vec{v}_1 \cdot \vec{v}_2,\ r_1\vec{v}_2 + r_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2)$

4.1 Quaternions as Rotations

We start with a vector, $\vec{p} = (p_x, p_y, p_z)$, written as a quaternion with real coordinate zero,

$$p = p_x\mathbf{i} + p_y\mathbf{j} + p_z\mathbf{k}$$

We want a rotation of p through an angle of θ about the axis defined by a unit vector

$$\vec{u} = (u_x, u_y, u_z) = u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k}$$

In two dimensions, Euler's Formula, $e^{i\theta} = \cos\theta + i\sin\theta$, gives a counterclockwise rotation of θ. We can extend it to three dimensions as

$$q = e^{\frac{\theta}{2}(u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})} = \cos\frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2}$$

The rotation of $\vec{p}$ about $\vec{u}$ is given by

$$p' = qpq^{-1}$$

Going back to using Euler's Formula for a rotation, let's look at the multiplicative inverse of q and see that it's consistent with previous knowledge. By a previous formula,

$$a^{-1} = (a_w + a_x\mathbf{i} + a_y\mathbf{j} + a_z\mathbf{k})^{-1} = \frac{1}{a_w^2 + a_x^2 + a_y^2 + a_z^2}(a_w - a_x\mathbf{i} - a_y\mathbf{j} - a_z\mathbf{k})$$

The vector $\vec{u} = u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k}$ is a unit vector with real part zero, so the denominator is 1.

$$(\vec{u})^{-1} = (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})^{-1} = -u_x\mathbf{i} - u_y\mathbf{j} - u_z\mathbf{k} = -\vec{u}$$

Applying the extension of Euler's Formula,

$$q = e^{\frac{\theta}{2}(u_x\mathbf{i}+u_y\mathbf{j}+u_z\mathbf{k})} = \cos\frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2}$$

and the inverse formula

$$a^{-1} = (a_w + a_x\mathbf{i} + a_y\mathbf{j} + a_z\mathbf{k})^{-1} = \frac{1}{a_w^2 + a_x^2 + a_y^2 + a_z^2}(a_w - a_x\mathbf{i} - a_y\mathbf{j} - a_z\mathbf{k})$$

we get

$$\begin{aligned} q^{-1} &= \frac{1}{\cos^2\frac{\theta}{2} + u_x^2\sin^2\frac{\theta}{2} + u_y^2\sin^2\frac{\theta}{2} + u_z^2\sin^2\frac{\theta}{2}}\left(\cos\frac{\theta}{2} - (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2}\right) \\ &= \frac{1}{\cos^2\frac{\theta}{2} + (u_x^2 + u_y^2 + u_z^2)\sin^2\frac{\theta}{2}}\left(\cos\left(-\frac{\theta}{2}\right) + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\left(-\frac{\theta}{2}\right)\right) \\ &= \frac{1}{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}}\left(\cos\left(-\frac{\theta}{2}\right) + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\left(-\frac{\theta}{2}\right)\right) \\ &= e^{\frac{\theta}{2}(-u)} = \left(e^{\frac{\theta}{2}u}\right)^{-1} \end{aligned}$$

5 Thursday 23 August: Raster Display v/s Vector Display

Ray Tracing: Which objects intersect the line?

Polygon Projection (Polygon Rendering)
Triangles on the screen
Vertex coordinates
Multiply by matrices to convert to screen's coordinate system

Modeling Coordinates
→ World Coordinates
→ Viewing and Projection Coordinates (Camera View)

→ Normalized Coordinates

→ Device Coordinates

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \text{Model Martrix} \rightarrow \text{Projection Matrix (Camera View)} \rightarrow \text{Perspective Division} \rightarrow \text{Viewpoint Transformation}$$

2D Scan Conversion: Converts 2D object description to pixel values.
"Pixel" Picture Element
Visible Surface Determination: Occlusion

Direct Illumination v/s Global Illumination: Shadows not completely dark
Curves and surfaces give a smooth alternative to polygonal representation of objects.

Lots of student projects have dealt with **Particle Dynamics**

Early Video Games Nimatron (1940) First "video" game used an oscilloscope as its screen.

Raster Display - Image broken into pixels
Vector Display - Smooth curves, like moving lasers or electron beam.

Black & white **bitmap** has one bit per pixel

Raster Displays
More computationally expensive
Requires more memory
Constant refresh rate
Supports area fills
Won over vector after memory got cheap.

Frame buffers are getting more complex.

- Double buffering so screen doesn't refresh in the middle of a memory move
- Left and right buffers for stereoscopic rendering
- Depth buffer for occlusion

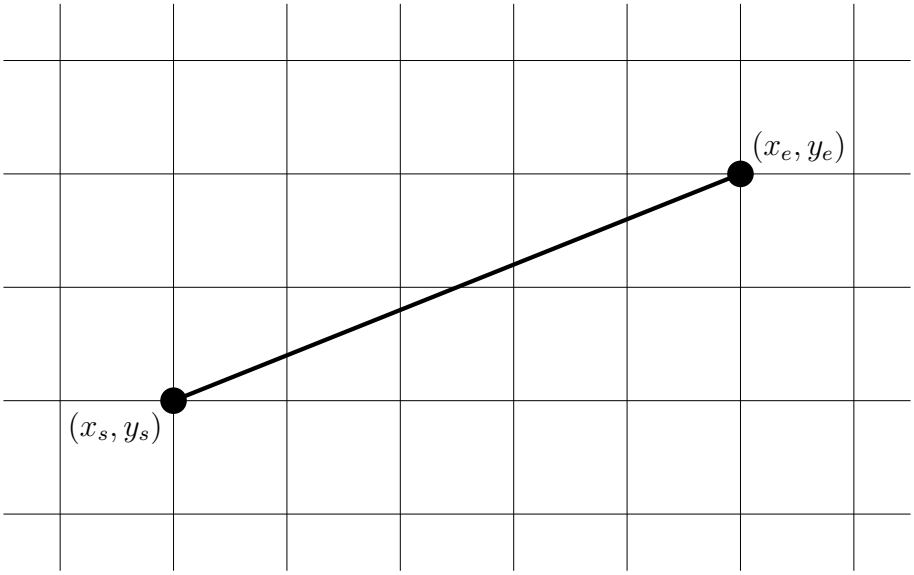
Know from Today's Lesson
Pixel, raster, bitmap, frame buffer, *aliasing*

Know the difference between raster and vector.

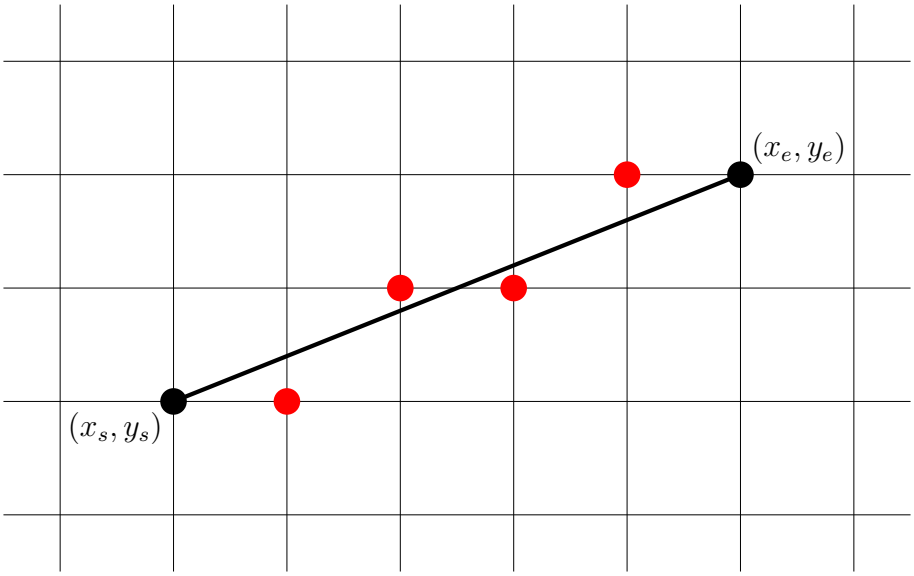
Aliasing - in computer graphics, the jagged, or saw-toothed appearance of curved or diagonal lines on a low-resolution monitor.

6 Tuesday 28 August: Scan Conversion of a Line Segment

Simplifying Assumptions
1-pixel-thick line on a B&W display (monochrome)
$m \in [-1, 1]$. More elaborate code can deal with other slopes.
Line segment described by two integer endpoints (endpoints fall exactly on pixels) $(x_s, y_s), (x_e, y_e)$



Main Idea: Activate one pixel per column from x_s to x_e.



Simple, but Slow, Algorithm

$m = (y_e - y_s)/(x_e - x_s)$

$b = y_s - m \cdot x_s$

for x from x_s to x_e *(inclusive)*

    color pixel at $(x, \lfloor m \cdot x + b + 0.5 \rfloor)$

The problem with this algorithm that is the $m \cdot x$, floating-point multiplication, is really computationally expensive, usually six cycles, while addition is relatively cheap, usually one cycle.

Rendering algorithms have to be as fast as possible, because they run billions of times.

Basic Incremental Algorithm (DDA, Digital Differential Analyzer)

$m = (y_e - y_s)/(x_e - x_s)$

$b = y_s - m \cdot x_s$

$x_0 = x_s$

$y_0 = y_s$

for i from 1 to $(x_e - x_s)$

    $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i + m)$

    color pixel at $(x_i + 1, \lfloor y_{i+1} + 0.5 \rfloor)$

Midpoint Algorithm (Besenham)

Developed for pen plotter.

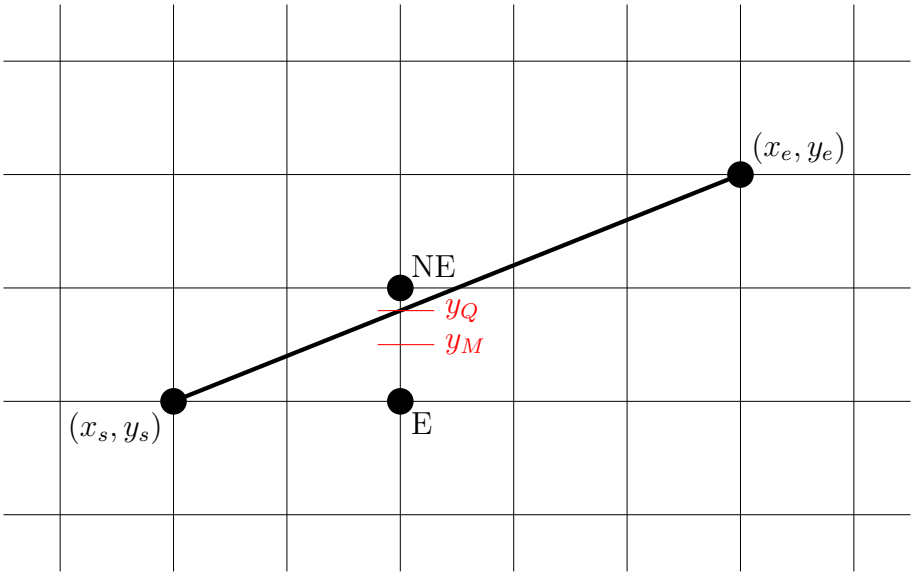
Bottleneck was algorithm and computational speed, not hardware speed.

Simplifying assumption: $m \in [0, 1]$

Main idea

Move either E or NE in each move.

Choose based on value of decision variable, d, which lets us choose betwen E and NE.



Notation: Q for *crossing*, M for *midpoint*.

Algorithm

$d = y_Q - y_M$.

Look at sign of d.

Move NE when d is positive.

Move E otherwise.

Computing d

Initialize $d = m - 0.5$

Increment:

for E moves: $d = d + m$

for NE moves: $d = d + m - 1$

Here's how Midpoint is cheaper than DDA: We can change everything to integers and not have any floats.

Make Everything Integers

Scale everything by $2\Delta x$, so that $m - 0.5$ becomes an integer.
$d = 2\Delta y - \Delta x$
East increment: $d = d + 2\Delta y$
NE increment: $d = d + 2\Delta y - 2\Delta x$

These methods aren't what we use today. Probably triangle scan conversion.

7 Thursday 30 August: Scan Conversion of a Triangle

7.1 Homework 1

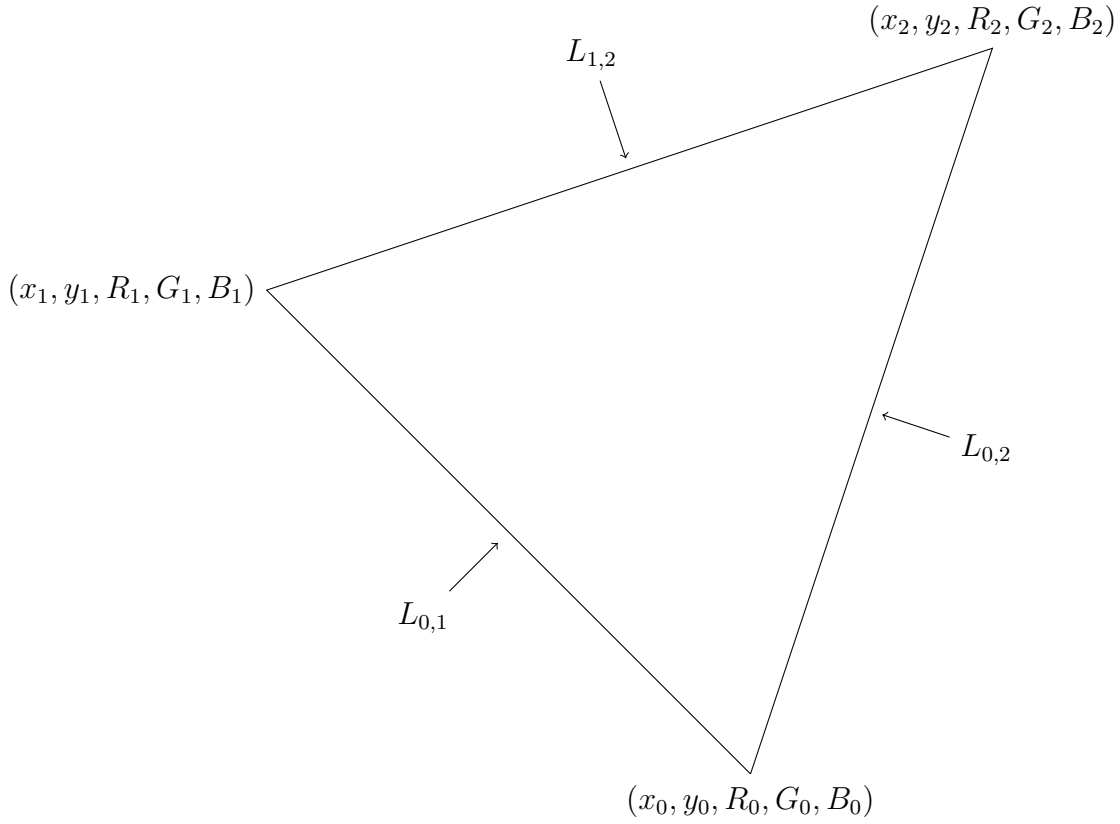
due 13 September
Callback Do sth when sth happens
Will have global variables
Assignment is given as a triangle with a certain order of x_0, x_1, and x_2. Extra part of assignment is to account for different orders.

7.2 Midpoint Algorithm (Besenham) Pseudocode

Initialize integers Δ_E, Δ_{NE}, d, x, and y.
Set pixel at (x, y) (first pixel)

while $x <$ last column (given by an endpoint)
{
    increment x by 1
    if $(d < 0)$
        add Δ_E to d
    else
    {
        increment y by 1
        add Δ_{NE} to d
    }
    set pixel at (x, y)
}

7.3 Naming Conventions



Simplifying Assumptions:

Vertices have integer coordinates

$y_0 \leq y_1 \leq y_2$

Idea

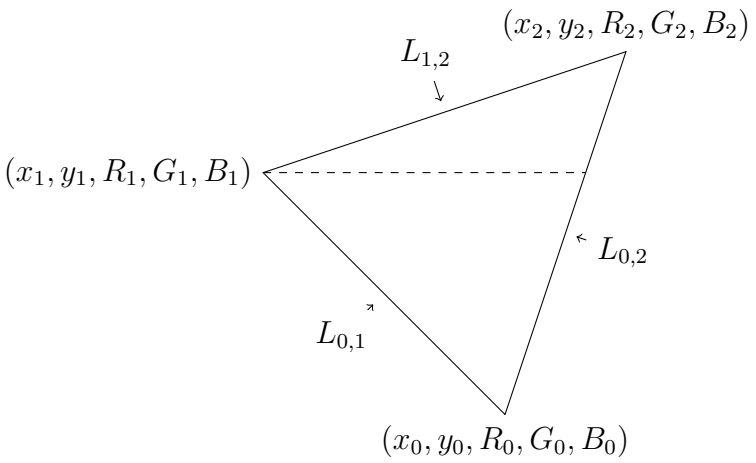
We want to interpolate the colors to get shading.

Look at the triangle one scan line at a time, looping bottom to top, left to right.

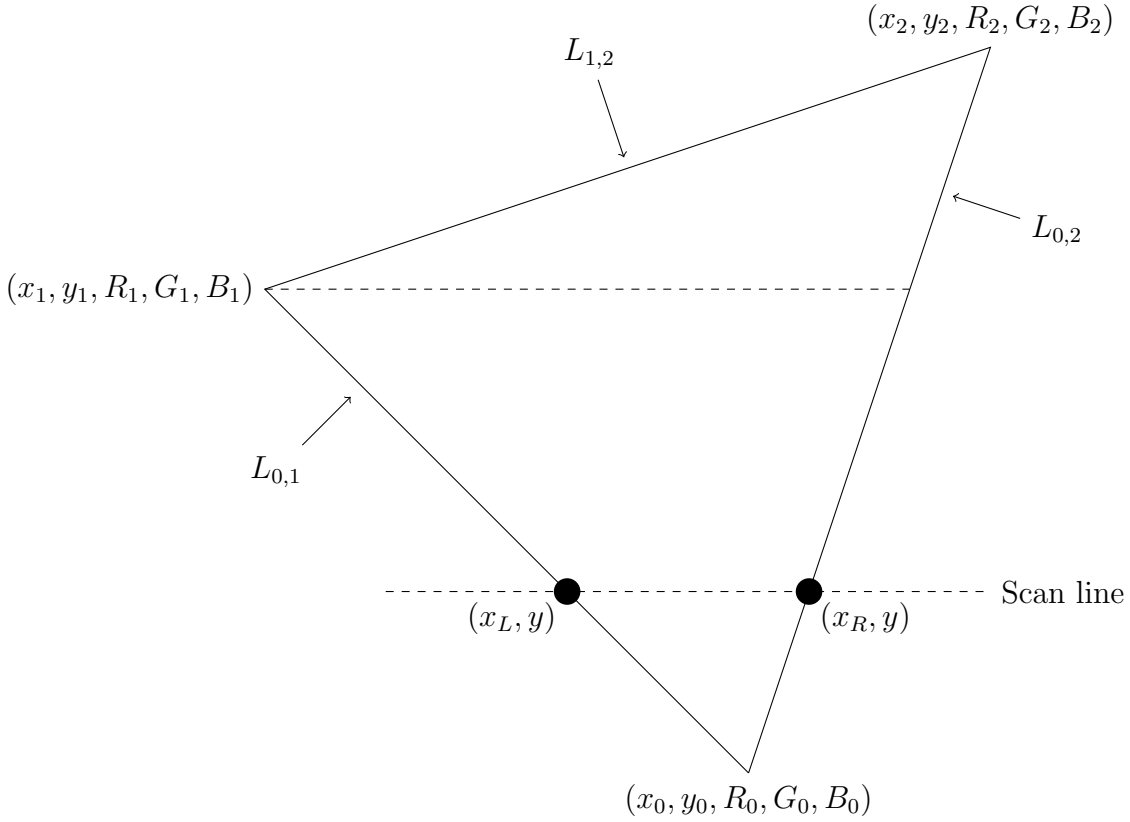
Two loops

One for the bottom

One for the top



Scan Line



7.4 Line Scan Pseudocode

for y from y_0 to $(y_1 - 1)$
{
    Calculate coordinates where scan line intersects $L_{0,1}$ and $L_{0,2}$.

(*i.e.* Calculate x_L and x_R incrementally)

Color the pixels from $(\lceil x_L \rceil, y)$ to $(\lfloor x_R \rfloor, y)$

}

for y from y_1 to y_2

{

Calculate coordinates where scan line intersects $L_{1,2}$ and $L_{0,2}$.

(*i.e.* Calculate x_L and x_R incrementally)

Color the pixels from $(\lceil x_L \rceil, y)$ to $(\lfloor x_R \rfloor, y)$

}

7.5 Special Cases

Watch for division by zero!

Horizontal lines, either between V_0 and V_1 or between V_1 and V_2.

7.6 Calculating x_L and x_R Incrementally

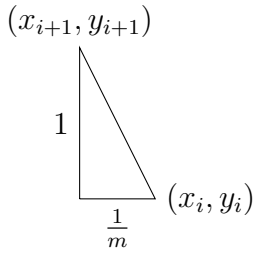
Initialize $x_L = x_0$, $x_R = x_0$, $y = y_0$

Note that y is being incremented by 1.

$$x_{i+1} = x_i + \frac{1}{m} = x_i + \frac{x_1 - x_0}{y_1 - y_0}$$

$$x_L = x_L + + \frac{x_1 - x_0}{y_1 - y_0}$$

$$x_R = x_R + + \frac{x_2 - x_0}{y_2 - y_0}$$



7.7 Color

Notes on this section:

Make sure we're consistent about when to use $x_L - x_R$ and when to use $\lceil x_L \rceil - \lfloor x_R \rfloor$.

Computing color incrementally (Here considering only R, red)

Initialize $R_L = R_0$, $R_R = R_0$

For each scan line:

$$R_L = R_L + \frac{R_1 - R_0}{y_1 - y_0}$$

$$R_R = R_R + \frac{R_2 - R_0}{y_2 - y_0}$$

Within a scan line, between $\lceil x_L \rceil$ and $\lfloor x_R \rfloor$

Initialize $R = R_L + \dfrac{R_L - R_R}{x_L - x_R}(\lceil x_L \rceil - x_L)$
Color first pixel
for R from R_L to R_R
{
    $R = R + \dfrac{R_L - R_R}{\lceil x_L \rceil - \lfloor x_R \rfloor}$
    [Do the same for G and B.]
    Color pixel (x, y, R, G, B)
}