A Novel, Symmetrical Solution to Intersection of 2 Lines
by Arav Bhattacharya (Dataflow Geometry Tour Guide A)

[Video version: https://youtu.be/UbVjSkA18D0?si=SGWtxGlnH24sjQuY&t=2342]

A mid-level problem in *DataflowGeometry2D* is finding the intersection of two lines. In this section, we describe one way that this problem can be solved. In *DataflowGeometry2D*, lines are represented numerically via *orientation* $\mathbf{o}$ and *location* l (explained here). Therefore, to find the intersection of two lines, we must use their respective orientations ($\mathbf{o}_1$ and $\mathbf{o}_2$) and locations (l_1 and l_2) to find their intersection point $\mathbf{i}$, as shown in Figure 1a below.

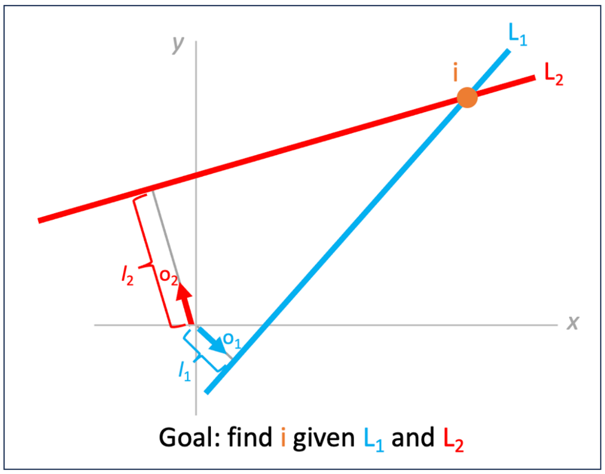

Fig. 1a. Sketch of problem statement illustrating given inputs

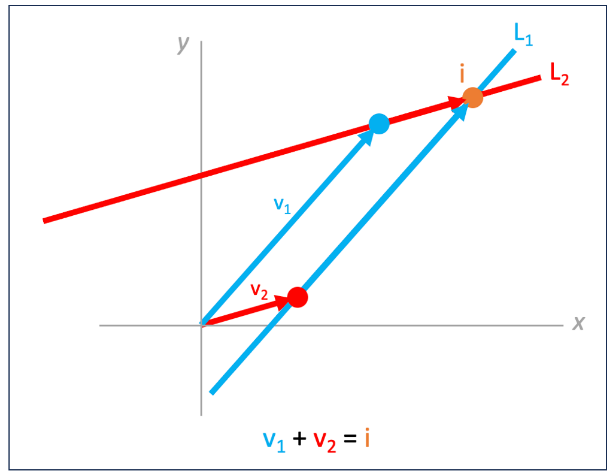

Fig. 1b. High-level solution

The problem-solving methodology relies heavily on visual imagination and sketching. In that mode, one can visualize a parallelogram spanned by two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ *drawn parallel to* $\mathbf{L}_1$ and $\mathbf{L}_2$ (Fig. 1b). There a straighforward solution suggested by this parallelogram.

The parallelogram so constructed implies that:

$$\mathbf{i} = \mathbf{v}_1 + \mathbf{v}_2$$

Can we solve independently for each of the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ and then add them together to get our final solution $\mathbf{i}$? Yes. Furthermore, there is a natural symmetry to this problem in that the process of solving for $\mathbf{v}_2$ is identical to the process of solving for $\mathbf{v}_1$ after swapping variables. This means that if we can solve for $\mathbf{v}_1$, then we can use the same process to find $\mathbf{v}_2$.

To solve for $\mathbf{v}_1$, we need its direction and magnitude. By definition, $\mathbf{v}_1$ runs parallel to $\mathbf{L}_1$, so $\mathbf{dir}(\mathbf{v}_1)$ is the same as the run direction of $\mathbf{L}_1$. This direction is also the same direction as a 90° counterclockwise rotation applied to $\mathbf{L}_1$ orientation $\mathbf{o}_1$. This 90° vector rotation may be computed using a previously-solved *DataflowGeometry2D* module.

We have solved for the direction of $\mathbf{v}_1$. How can we obtain magnitude $\| v_1 \|$? Figure 2 gives the strategy.

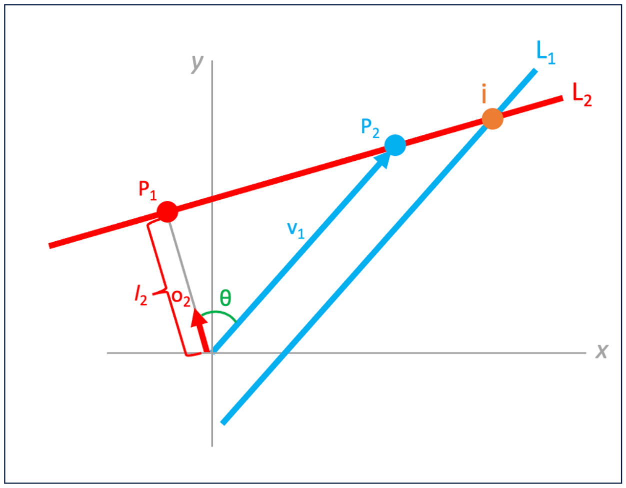

Fig. 2. Right triangle that solves for $\| v_1 \|$

Let's refer to the angle between $\mathbf{o}_2$ and $\mathbf{v}_1$ as θ. Since orientation vector $\mathbf{o}_2$ by definition makes a right angle with $\mathbf{L}_2$, the triangle $\mathbf{OP_1P_2}$ (where $\mathbf{O}$ is the origin) is a right triangle. As a result:

$$\cos\theta = \frac{l_2}{|| v_1 ||} \qquad (\cos\theta = \text{adjacent / hypoteneuse})$$

$$|| v_1 || = \frac{l_2}{\cos\theta} \qquad (\text{rearrange to solve for } || v_1 ||)$$

$$\cos\theta = \mathrm{dir}(v_1) \cdot \mathbf{o_2} \qquad (\text{cosine of angle bracketed by two unit vectors is their dot product})$$

$$|| v_1 || = \frac{l_2}{\mathrm{dir}(v_1) \cdot \mathbf{o_2}} \qquad (\text{substitute dot product for } \cos\theta)$$

We finish solving $\mathbf{v}_1$ by combining its magnitude and direction (by scalar multiplication):

$$\mathbf{v_1} = || v_1 || \ \mathbf{dir\,(v_1)} = \left(\frac{l_2}{\mathrm{dir}(v_1) \cdot \mathbf{o_2}} \right) \mathbf{dir\,(v_1)}$$

and by the exact same reasoning, arrive at a symmetrical solution for $\mathbf{v}_2$:

$$\mathbf{v_2} = || v_2 || \ \mathbf{dir\,(v_2)} = \left(\frac{l_1}{\mathrm{dir}(v_2) \cdot \mathbf{o_1}} \right) \mathbf{dir\,(v_2)}$$

Note that in a case like this, $\mathbf{dir(v_2)}$ points away from the intersection; however, whenever this happens in our solution, $\mathbf{dir(v_2)} \cdot \mathbf{o_1}$ is negative and hence $||v_2||$ is negative, thus $\mathbf{v}_2$ comes out with the correct pointing direction.

The above two expressions specify how to calculate $\mathbf{v}_1$ and $\mathbf{v}_2$ from the givens L1 = [$\mathbf{o}_1$ l_1] and L2 = [$\mathbf{o}_2$ l_2]. We translate these expressions into dataflows (Figure 3), merging them using vector addition ($\mathbf{i} = \mathbf{v}_1 + \mathbf{v}_2$) to compute the final numerical result for the intersection point.

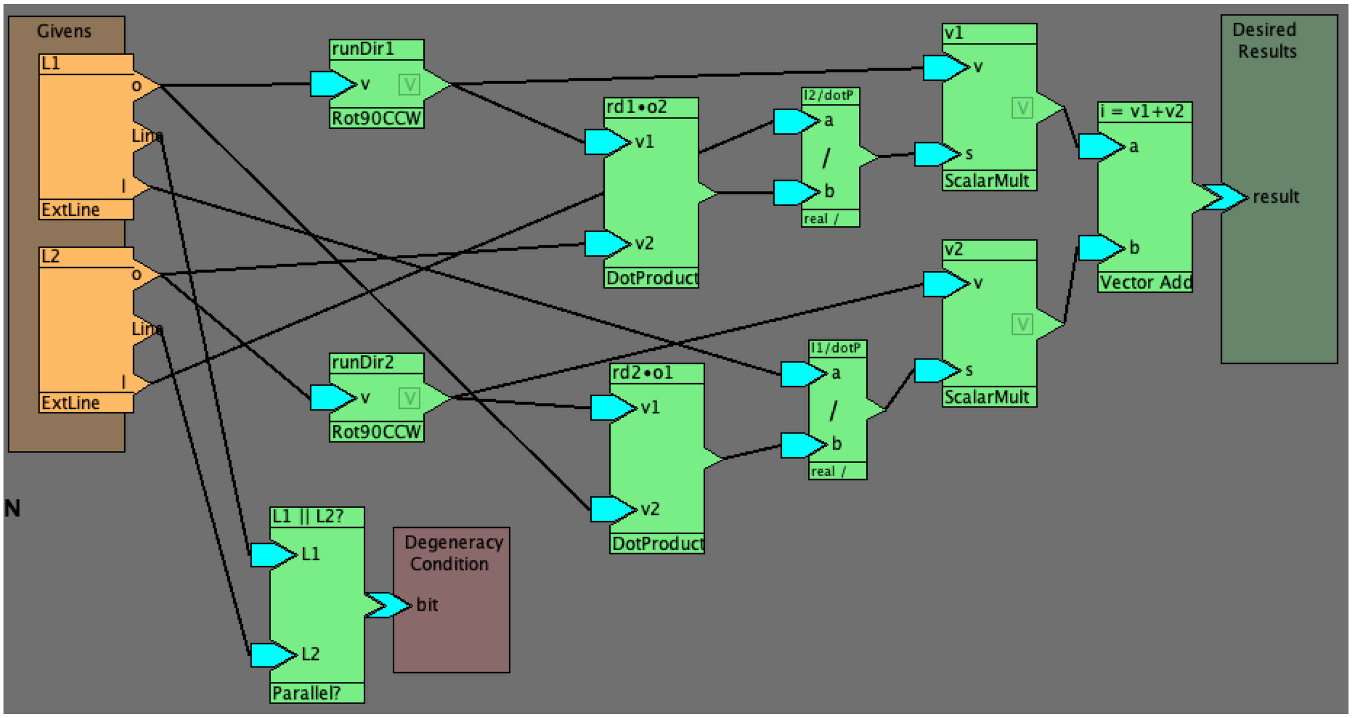

Figure 3. Modular Dataflow implementing Bhattacharya's Symmetrical Computational Solution for the Intersection of any two Given Lines L1 & L2

The last issue to tackle is degeneracy. Two lines will have a degenerate intersection if they are parallel. How does this impact the calculations? The II v_1 II and II v_2 II calculations blow up (division by 0, arising from the dot product of 2 orthogonal vectors). Therefore, detecting this special case and reporting it adds a measure of robustness to the automated solution being created. The previously-solved module `parallel?(L1,L2)` is patched in as the desired detector. With degeneracy reporting, the modularized line intersection algorithm becomes fault-tolerant in any usage context. This feature supports drama-free solution piggybacking over arbitrary levels. A whole slew of more advanced problems that require intersecting two lines can now be undertaken, e.g., solving a circle's unknown center and radius given 3 points on the circle.

We conclude with a few words about this solution. Since most students will be used to the algebraic representations of lines taught in Algebra I (standard, slope-intercept, or point-slope form), [**o** *l*] representation provides students a dash of novelty to a problem that they will no doubt have solved before, but this time with robust automation of their mathematical thinking.

Additionally, the solution we demonstrate was derived using simple vector functions and trigonometry, but can also be derived with methods ranging from freshman algebra to college linear algebra; thus, many high school students can approach, enjoy, and learn from this problem, regardless of their age and mathematical background.