The Maths !

In order to perform ray tracing, we need to be able to detect intersections between rays and triangles

We can represent a ray using a start point 's'
And a direction vector 'd'

Any position along that ray can be represented as:

position = startpoint + scalar * direction

or

- `r = s + t * d`

Triangular Plane

The equation that defines points on a triangular plane in 3D space is bit more tricky !

As always, let's start by considering the topic in 2D
Then move on to the more complex 3D situation

PointOnTriangle

Solving

Taking:

$$r = p_0 + u(p_1-p_0) + v(p_2-p_0)$$

We can simplify it by replacing points with edges:

$$r = p_0 + ue_0 + ve_1$$

Then combine this with previous equation for a ray:

$$r = s + t * d$$

To get:

$$p_0 + ue_0 + ve_1 = s + t * d$$

Rearrange to get:

$$- t * d + ue_0 + ve_1 = s - p_0$$

Matrix Form

Everything we do is in matrix form, so rewrite:

```
- t * d + ue₀ + ve₁ = s - p₀
```

as:

$$\begin{bmatrix} -d^x & e_0^x & e_1^x \\ -d^y & e_0^y & e_1^y \\ -d^z & e_0^z & e_1^z \end{bmatrix} \cdot \begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} s^x - P_0^x \\ s^y - P_0^y \\ s^z - P_0^z \end{bmatrix}$$

Rearrange

Rearranging we end up with...

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} -d^x & e_0^x & e_1^x \\ -d^y & e_0^y & e_1^y \\ -d^z & e_0^z & e_1^z \end{bmatrix}^{-1} \cdot \begin{bmatrix} s^x - P_0^x \\ s^y - P_0^y \\ s^z - P_0^z \end{bmatrix}$$

And that's it !

t, u, v are what we are looking for
(distance along ray and the coords on the triangle)

The 3x3 "d e" matrix can easily be constructed
(by pulling out elements of different vec3 variables)

The "S P" vector can also easily be constructed
(perhaps even more easily than you think !)

There is even an inverse function in the GLM library
(so you don't have to write one !)