# An Efficient Ray-Triangle Intersection Algorithm

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

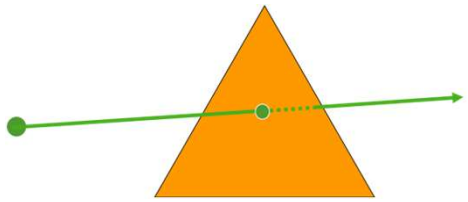


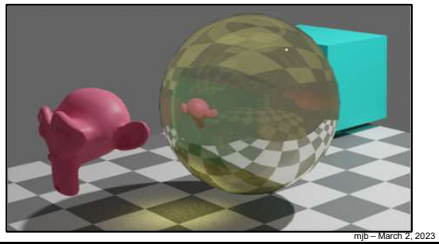




## Why Do We Want to Intersect a Ray and a Triangle?

There are many applications for finding if a line intersects the inside of a triangle, and, if so, where. Examples include collision detection, ray-tracing, etc.





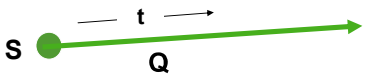




## Parametrizing a Ray

**Given:**
**S** is the (x,y,z) starting point
**Q** is the (x,y,z) direction of travel

Then, the (x,y,z) position of a point ***p*** at some position along its direction of travel is:




$$p = S + tQ$$

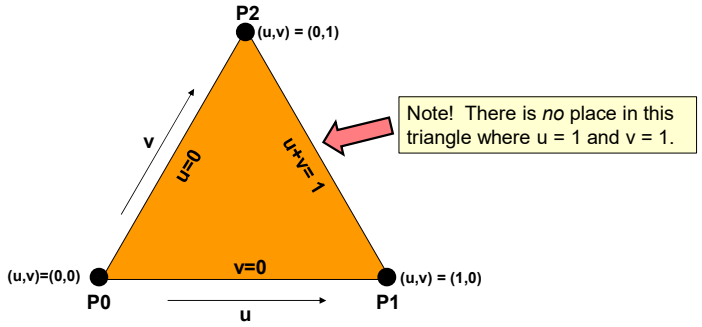
$t \geq 0.$





## Parametrizing a Triangle

It's often useful to be able to parameterize a triangle into (u,v), like this:



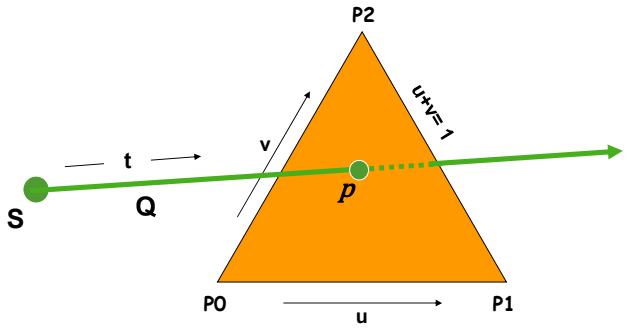

**p = P0 + u*(P1-P0) + v*(P2-P0)**

## The Setup



We want to find out where the ray intersects the triangle.

That is, where is the point $\boldsymbol{p}$ that is common to both the ray and the triangle?




Such that:

$$t \geq 0.$$
$$0. \leq u \leq 1.$$
$$0. \leq v \leq 1.\text{-u}$$



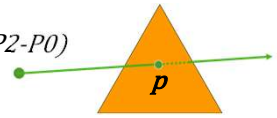
5

## Equation Setup



Triangle: $p = P0 + u*(P1\text{-}P0) + v*(P2\text{-}P0)$

Ray: $p = S + tQ$




Re-arranging:

$$P0 + u*(P1\text{-}P0) + v*(P2\text{-}P0) = S + tQ$$

Re-arranging some more:

$$-tQ + u*(P1\text{-}P0) + v*(P2\text{-}P0) = S - P0$$

Then collecting terms, we get:

$$At + Bu + Cv = D$$

where:

$$A = -Q$$
$$B = P1\text{-}P0$$
$$C = P2\text{-}P0$$
$$D = S - P0$$



6

## Three Equations, Three Unknowns



Remembering that this equation is really 3 equations in (x,y,z):

$$At + Bu + Cv = D$$

we have 3 equations with 3 unknowns, which can be cast into a matrix form

$$\begin{bmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ A_z & B_z & C_z \end{bmatrix} \begin{Bmatrix} t \\ u \\ v \end{Bmatrix} = \begin{Bmatrix} D_x \\ D_y \\ D_z \end{Bmatrix}$$

Our goal is to solve this for t*, u*, and v*



7

## Solve for (t*,u*,v*) using Cramer's Rule



$$\begin{bmatrix} A_x & B_x & C_x \\ A_z & B_y & C_y \\ A_z & B_z & C_z \end{bmatrix} \begin{Bmatrix} t \\ u \\ v \end{Bmatrix} = \begin{Bmatrix} D_x \\ D_y \\ D_z \end{Bmatrix}$$

$$D_0 = det \begin{bmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ A_z & B_z & C_z \end{bmatrix}$$

$$D_t = det \begin{bmatrix} D_x & B_x & C_x \\ D_y & B_y & C_y \\ D_z & B_z & C_z \end{bmatrix} \qquad \boldsymbol{t^* = \frac{D_t}{D_0}}$$

$$D_u = det \begin{bmatrix} A_x & D_x & C_x \\ A_y & D_y & C_y \\ A_z & D_z & C_z \end{bmatrix} \qquad \boldsymbol{u^* = \frac{D_u}{D_0}}$$

$$D_v = det \begin{bmatrix} A_x & B_x & D_x \\ A_y & B_y & D_y \\ A_z & B_z & D_z \end{bmatrix} \qquad \boldsymbol{v^* = \frac{D_v}{D_0}}$$



8

## Flashback: The Determinant of a 3x3 Matrix



$$det \begin{bmatrix} M_{00} & M_{01} & M_{02} \\ M_{10} & M_{11} & M_{12} \\ M_{20} & M_{21} & M_{22} \end{bmatrix} =$$

$$\boldsymbol{M_{00}} * [M_{11} * M_{22} - M_{21} * M_{12}] \; \boldsymbol{- M_{01}} * [M_{10} * M_{22} - M_{20} * M_{12}] + \boldsymbol{M_{02}} * [M_{10} * M_{21} - M_{20} * M_{11}]$$



## The Steps



1. Compute $D_0$
2. If $D_0 \approx 0.$, then the ray is *parallel* to the plane of the triangle 

3. Compute $D_t$
4. Compute t*
5. If t* < 0., the ray goes away from the triangle 

6. Compute $D_u$
7. Compute u*
8. If u* < 0. or u* > 1., then the ray hits outside the triangle 

9. Compute $D_v$
10. Compute v*
11. If v* < 0. or v* > 1.-u*, then the ray hits outside the triangle 

12. The intersection is at the point $\boldsymbol{p = S + Qt^*}$

This is known as the **Möller-Trumbore Triangle Intersection Algorithm**



## Computing the Determinant of a 3-Column Matrix using GLM



```
float
Determinant( glm::vec3 c0, glm::vec3 c1, glm::vec3 c2 )
{
        float d00 = c0.x * ( c1.y*c2.z – c1.z*c2.y );
        float d01 = c1.x * ( c0.y*c2.z – c0.z*c2.y );
        float d02 = c2.x * ( c0.y*c1.z – c0.z*c1.y );
        return d00 – d01 + d02;
}
```



## Setting Up the Equations



```
float Ax = -Qx;
float Ay = -Qy;
float Az = -Qz;

float Bx = P1x – P0x;
float Bx = P1y – P0y;
float Bx = P1z – P0z;

float Cx = P2x – P0x;
float Cx = P2y – P0y;
float Cx = P2z – P0z;

float Dx = Sx – P0x;
float Dx = Sy – P0y;
float Dx = Sz – P0z;
```

## Cramer's Rule using GLM

```
glm::vec3 colA  = glm::vec3( Ax, Ay, Az );
glm::vec3 colB  = glm::vec3( Bx, By, Bz );
glm::vec3 colC  = glm::vec3( Cx, Cy, Cz );
glm::vec3 colD  = glm::vec3( Dx, Dy, Dz );

float d0 = Determinant( colA, colB, colC );
float dt  = Determinant( colD, colB, colC );
float du = Determinant( colA, colD, colC );
float dv = Determinant( colA, colB, colD );

float tstar  = dt  / d0;
float ustar = du / d0;
float vstar = dv / d0;
```

Oregon State University
Computer Graphics