An Efficient Ray-Triangle Intersection Algorithm

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.

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

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:

\[ (u,v) = (0,0) \]
\[ (u,v) = (0,1) \]
\[ (u,v) = (1,0) \]

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

Note! There is no place in this triangle where \( u = 1 \) and \( v = 1 \).

The Setup

We want to find out where the ray intersects the triangle. That is, where is the point \( p \) that is common to both the ray and the triangle?

\[ t \geq 0. \]
\[ 0 \leq u \leq 1. \]
\[ 0 \leq v \leq 1-u. \]

Equation Setup

Triangle: \( p = P0 + u*(P1-P0) + v*(P2-P0) \)
Ray: \( p = S + tQ \)

Re-arranging:
\[ P0 + u*(P1-P0) + v*(P2-P0) = S + tQ \]

Re-arranging some more:
\[ -Q + u*(P1-P0) + v*(P2-P0) = S - P0 \]

Then collecting terms, we get:
\[ At + Bu + Cv = D \]

where:
\[ A = -Q \]
\[ B = P1-P0 \]
\[ C = P2-P0 \]
\[ D = S - P0 \]
Three Equations, Three Unknowns

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

\[ A t + B u + C v = D \]

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 \( v^* \)

Solve for \( (t^*,u^*,v^*) \) using Cramer’s Rule

\[
D_0 = \text{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 = \text{det}
\begin{bmatrix}
B_x & C_x & D_x \\
B_y & C_y & D_y \\
B_z & C_z & D_z \\
\end{bmatrix}

D_u = \text{det}
\begin{bmatrix}
A_x & B_x & D_x \\
A_y & B_y & D_y \\
A_z & B_z & D_z \\
\end{bmatrix}

D_v = \text{det}
\begin{bmatrix}
A_x & B_y & C_x \\
A_y & B_y & C_y \\
A_z & B_z & C_z \\
\end{bmatrix}

\]

\[ t^* = \frac{D_t}{D_0} \]

\[ u^* = \frac{D_u}{D_0} \]

\[ v^* = \frac{D_v}{D_0} \]

Flashback: The Determinant of a 3x3 Matrix

\[
\text{det}
\begin{bmatrix}
M_{00} & M_{01} & M_{02} \\
M_{10} & M_{11} & M_{12} \\
M_{20} & M_{21} & M_{22} \\
\end{bmatrix}
= 
M_{00} \cdot [M_{11} \cdot M_{22} - M_{12} \cdot M_{21}] - M_{01} \cdot [M_{10} \cdot M_{22} - M_{12} \cdot M_{20}] + M_{02} \cdot [M_{10} \cdot M_{21} - M_{11} \cdot M_{20}]
\]

Setting Up the Equations

\[
\begin{align*}
Ax &= -Qx; \\
Ay &= -Qy; \\
Az &= -Qz; \\
Bx &= P1x - P0x; \\
By &= P1y - P0y; \\
Bz &= P1z - P0z; \\
Cx &= P2x - P0x; \\
Cy &= P2y - P0y; \\
Cz &= P2z - P0z; \\
Dx &= Sx - P0x; \\
Dy &= Sy - P0y; \\
Dz &= Sz - P0z; \\
\end{align*}
\]
Cramer's Rule using GLM

```cpp
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;
```