Vulkan Ray Tracing – 5 New Shader Types!

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

Analog Ray Tracing Example

Digital Ray Tracing Examples

In a Raytracing, each ray typically hits a lot of Things

Example: The Ray Intersection Process for a Sphere

S = 𝑆 + 𝑡𝑄
t ≥ 0.

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 ≥ 0.

Example: The Ray Intersection Process for a Sphere

Sphere equation: (𝑥−𝑐)² + (𝑦−𝑐)² + (𝑧−𝑐)² = 𝑅²
Ray equation: (𝑥,𝑦,𝑧) = (𝑥₀,𝑦₀,𝑧₀) + 𝑡(𝑑𝑥,𝑑𝑦,𝑑𝑧)

Plugging (𝑥,𝑦,𝑧) from the second equation into the first equation and multiplying-through and simplifying gives:

𝐴𝑡² + 𝐵𝑡 + 𝐶 = 0 t₁, t₂ = \frac{-𝐵 ± \sqrt{𝐵² − 4AC}}{2𝐴}

Solve for 𝑡₁, 𝑡₂ and analyze the solution like this:
1. If both 𝑡₁ and 𝑡₂ are complex (i.e., have an imaginary component), then the ray missed the sphere completely.
2. If both 𝑡₁ and 𝑡₂ are real and identical, then the ray brushed the sphere at a tangent point.
3. If both 𝑡₁ and 𝑡₂ are real and different, then the ray entered and exited the sphere.
Parameterizing a Triangle

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

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

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

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

\[
p = P_0 + u(P_1 - P_0) + v(P_2 - P_0)
\]

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

\[
p = P_0 + u(P_1 - P_0) + v(P_2 - P_0)
\]

\[
p = S + tQ
\]

\[
P_0 + u(P_1 - P_0) + v(P_2 - P_0) = S - P_0
\]

\[
A t + Bu + Cv = D
\]

\[
A = -Q
\]

\[
B = P_1 - P_0
\]

\[
C = P_2 - P_0
\]

\[
D = S - P_0
\]

Re-arranging:

\[
A t + Bu + Cv = D
\]

Re-arranging some more:

\[
-AQ + u(P_1 - P_0) + v(P_2 - P_0) = S - P_0
\]

Then collecting terms, we get:

\[
At + Bu + Cv = D
\]

where:

\[
A = -Q
\]

\[
B = P_1 - P_0
\]

\[
C = P_2 - P_0
\]

\[
D = S - P_0
\]

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

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

\[
\begin{aligned}
D_0 &= \det
\begin{bmatrix}
A_x & B_x & C_x \\
A_y & B_y & C_y \\
A_z & B_z & C_z
\end{bmatrix}
\\
D_t &= \det
\begin{bmatrix}
A_x & B_x & C_x \\
A_y & B_y & C_y \\
A_t & B_t & C_t
\end{bmatrix}
\\
D_u &= \det
\begin{bmatrix}
A_x & B_x & C_x \\
A_y & B_y & C_y \\
A_u & B_u & C_u
\end{bmatrix}
\\
D_v &= \det
\begin{bmatrix}
A_x & B_x & C_x \\
A_y & B_y & C_y \\
A_v & B_v & C_v
\end{bmatrix}
\end{aligned}
\]

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

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

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

The Steps

1. Compute \(D_0\).
2. If \(D_0 \neq 0\), 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 \(p = S + Qt^*\).
The Rasterization Shader Pipeline That You Are used to Doesn’t Apply to Vulkan Ray Tracing

- Fixed Function
- Programmable

The Vulkan Ray Tracing Pipeline Involves Five New Shader Types

- Ray Generation Shader (rgen)
- Intersection Shader (rint)
- Any Hit Shader (rahit)
- Closest Hit Shader (rchit)
- Miss Shader (rmiss)

Unlike the rasterization pipeline, there is no constant flow from one shader to the next. Rather, particular shaders are called to respond to particular events.

- A Ray Generation Shader runs on a 2D grid of threads. It begins the entire ray-tracing operation.
- An Intersection Shader implements ray-primitive intersections.
- An Any Hit Shader is called when the Intersection Shader finds a hit. It decides if that intersection should be accepted or ignored.
- The Closest Hit Shader is called with the information about the hit that happened closest to the viewer. Typically, lighting is done here, or firing off new rays to handle shadows, reflections, and refractions.
- A Miss Shader is called when no intersections are found for a given ray. Typically, it just sets its pixel color to the background color.

Acceleration Structures

- A Bottom-level Acceleration Structure (BLAS) reads the vertex data from vertex and index VkBuffers to determine bounding boxes.
- You can also supply your own bounding box information to a BLAS.
- A Top-level Acceleration Structure (TLAS) holds transformations and pointers to multiple BLASes.
- The BLAS is essentially used as a Model Coordinate bounding box, while the TLAS is used as a World Coordinate bounding box.

Check This Out!

https://www.youtube.com/watch?v=QL7sXc2INJ8