Overview of Vulkan Ray Tracing “Pipeline”

Start With

- The usual transformation and projection matrices.
- Light locations.
- Geometry:
  - Triangles.
  - Shader-computable shapes. (E.g., true-sphere shader.)

Preparation

Prepare Acceleration Structures

These describe where geometry is.

Assemble a ray tracing "pipeline".

Start Ray Tracing (Each Frame)

Host records:

```
vk::CommandBuffer::traceRaysNV( shader_table, ...  width, height ..  );
```

Invokes the pipeline's Ray-Generation Shader width $\times$ height times.

A Ray

Like mathematical rays, has an origin (starting point) and a direction.

A starting and ending point: `t_min`, `t_max`.

For a mathematical ray `t_min = 0`, and `t_max = `∞.

But we are free to set `t_min = 0.1, t_max = 0.9`.

A payload. (A variable holding data shared by shaders.)

E.g., a color determined by what the ray intersected.

Flags. Specifies which shaders to use, etc.

E.g., `gl_RayFlagsOpaqueNV`. Don't bother with transparent geometry.

Cull Mask

An indicator of which geometry to consider.

Shader Stages – Quick Overview

Ray Generation Shader Stage:

The shader stage called from CPU code. Typically, each invocation casts a ray from the eye through a pixel. A ray payload, updated with a lighted color, is written to the frame buffer by the shader.

Closest Hit Shader Stage:

This shader is invoked during a ray cast and is given the geometry (triangle or AABB) closest to the ray origin.

Any Hit Shader Stage:

This shader is invoked during a ray cast and is given geometry (triangle or AABB) intersected by the ray. (One any-hit invocation for each item intersected.)

Intersection Shader Stage:

This shader is invoked during a ray cast and is given an AABB intersected by the ray.

Miss Shader Stage:

Called when a ray does not intersect anything.

Shader Data Access

Can access Uniform and Storage Buffers.

Ray Casting

Done by shaders. (Not host code.)

Cast by `traceNV( .., ray_origin, ..., ray_direction, ..  )`

Can be called by:

- Ray Generation Shaders
  - Sets `ray_origin` to the eye location.
  - Aims `ray_direction` at a pixel.
- Closest Hit Shaders, Any Hit Shaders
  - Sets `ray_origin` to hit location (position on geometry).
  - Might aim `ray_direction` at a light.
  - Might set `ray_direction` to reflected direction of incoming ray.

Calling traceNV starts Ray Traversal.

As a result of ray traversal shaders may be invoked. . .
. . . and those shaders may update the payload.

When traceNV returns the payload will have been updated by the invoked

shaders.

Ray Traversal

Calling traceNV starts Ray Traversal

Ray traversal is performed by the fixed functionality.

Ray traversal consists of finding geometry intersected by the ray.

- Ray traversal uses acceleration structures to guide the search.

- Intersection shaders are invoked during traversal.

Typically:

- The geometry-specific closest-hit shader is called for the geometry closest to the ray origin.

- Or the miss shader is called if no geometry is intersected.

Variations

Cull and ray flags can be used to limit geometry.

Typical Ray Generation Shader

Invoked for each pixel.

Casts a ray from eye to its assigned pixel.

Declares a ray payload that can carry a color.

- Determines eye location and its assigned pixel.

- Casts a ray at that pixel. (Calls traceNV)

- Shaders invoked by traceNV should set payload to a lighted color.

- After traceNV returns, the ray generation shader writes payload…
  … to frame buffer.

Closest and Any Hit Shaders

Built-In Variables

```
in     int    gl_PrimitiveID;          // Numbering of tri or AABBs within bottom accel structure.
in     int    gl_InstanceID;           // Numbering of instances specified in top accel structure.
in     int    gl_InstanceCustomIndexNV; // User's number assigned to instance.

in    vec3    gl_WorldRayOriginNV;     // Coordinate space of instance specified in top accel structure.
in    vec3    gl_WorldRayDirectionNV;
in    vec3    gl_ObjectRayOriginNV;    // Coordinate space of bottom accel structure.
in    vec3    gl_ObjectRayDirectionNV;

in    float   gl_HitTNV;               // Location of intersection.
in    uint    gl_HitKindNV;

in    mat4x3 gl_ObjectToWorldNV;       // I'd name it world_from_object.
in    mat4x3 gl_WorldToObjectNV;

in    uvec3   gl_LaunchIDNV;           // Pixel to be written.
in    uvec3   gl_LaunchSizeNV;         // Dimensions of window.

in    float   gl_RayTminNV,gl_RayTmaxNV; // Ray bounds.
in    uint    gl_IncomingRayFlagsNV;   // Flags given when ray cast.
```

Other Variables

Ray Payload:
Shared between shader casting a ray and shaders invoked due to the cast ray.

Hit Attribute:
Information about intersection. Barycentric coordinates for triangle intersections and custom data for intersection shaders.

Ray Payload

Data carried by a ray, such as a color.

Sample declarations:

```
layout ( location = 0 ) rayPayloadNV vec3 rp_color;
layout ( location = 0 ) rayPayloadInNV vec3 rp_color;
```

Qualifiers

`rayPayloadNV`: Used by shader that casts the ray.

`rayPayloadInNV`: Used by shader invoked due to the ray.

Just one payload variable per ray.

Locations

The location is used in the traceNV call.

Hit Attribute

Data describing the intersection.

For triangles (geometry not using an intersection shader):

Contains barycentric coordinates of intersection.

Example use for triangles:

```
hitAttributeNV vec2 bcoor;   // Declare hit attribute.

void main() // A Hit Shader
{
  vec4 c0 = my_color_list[ 3*gl_PrimitiveID ];
  vec4 c1 = my_color_list[ 3*gl_PrimitiveID + 1 ];
  vec4 c2 = my_color_list[ 3*gl_PrimitiveID + 2 ];

  vec4 blended_color = ( 1.0 - bcoor.x - bcoor.y ) * c0 + bcoor.x * c1 + bcoor.y * c2;
```

Using Hit Shader Built In Variables

Compute coordinates of intersected point.

```
void main() {  // A Hit Shader
  vec3 vertex_g = gl_WorldRayOriginNV + gl_WorldRayDirectionNV * gl_HitTNV;
  vec3 vertex_o = gl_ObjectRayOriginNV + gl_ObjectRayDirectionNV * gl_HitTNV;

  // Load data about this instance from a uniform array, uni_per_instance.
  RT_Uni_Per_Instance upi = uni_per_instance_a[gl_InstanceID];
  vec4 color = upi.color; // The same color is used in the entire instance.

  // Load data using custom-provided index.
  RT_Uni_Per_custom upci = uni_per_cinstance_a[gl_InstanceCustomIndexNV];
```