The NVIDIA® OptiX™ ray tracing engine

Austin Robison
The OptiX Engine

- A General Purpose Ray Tracing API
  - Rendering, baking, collision detection, A.I. queries, etc.
  - Modern “shader”-centric, stateless and bindless design
  - Is not a renderer but can implement many types of renderers

- Highly Programmable
  - Shading with arbitrary ray payloads
  - Ray generation/framebuffer operations (cameras, data unpacking, etc.)
  - Programmable intersection (triangles, NURBS, implicit surfaces, etc.)

- Easy to Program
  - Write single ray code (no exposed ray packets)
  - No need to rewrite shaders to target different hardware
Programmable Operations

<table>
<thead>
<tr>
<th>Rasterization</th>
<th>Ray Tracing</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Fragment</td>
<td>• Closest Hit</td>
</tr>
<tr>
<td>• Vertex</td>
<td>• Any Hit</td>
</tr>
<tr>
<td>• Geometry</td>
<td>• Intersection</td>
</tr>
<tr>
<td></td>
<td>• Selector</td>
</tr>
<tr>
<td></td>
<td>• Ray Generation</td>
</tr>
<tr>
<td></td>
<td>• Miss</td>
</tr>
<tr>
<td></td>
<td>• Exception</td>
</tr>
</tbody>
</table>

The ensemble of programs defines the rendering algorithm
(or collision detection algorithm, or sound propagation algorithm, etc.)
Entry Points
- Ray Generation Program
- Exception Program

rtTrace()

Ray Processing
- Closest Hit Program
- Miss Program

Traversal
- Intersection Program
- Any Hit Program
- Selector Visit Program

Buffers
- Texture Samplers
- Variables
- **Closest Hit Programs**: called once after traversal has found the closest intersection
  - Used for traditional surface shading
  - Deferred shading

- **Any Hit Programs**: called during traversal for each potentially closest intersection
  - Transparency without traversal restart (can read textures): `rtIgnoreIntersection()`
  - Terminate shadow rays that encounter opaque objects: `rtTerminateRay()`

- Both can be used for shading by modifying per ray state
Programming in OptiX

- Interconnection of programs defines the outcome
  - Whitted ray tracing, cook, path tracing, photon mapping
  - Or collision detection, sound propagation, ...
- Input “language” is based on CUDA C
  - No new language to learn
  - Powerful language features available immediately
    - Pointers
    - Templates
    - Overloading
    - Default arguments
    - Classes (no virtual functions)
  - Can also take raw PTX as input
- Caveat: still need to use it responsibly to get perf
Anatomy of a program

includes

declarations
  variables - program state (read only)
  textures - 1,2,3D (read only)
  buffers - 1,2,3D (read/write)

program definitions
  multiple allowed
Closest hit program (traditional “shader”)

- Defines what happens when a ray hits an object
- Executed for nearest intersection (closest hit) along a ray
- Automatically performs deferred shading
- Can recursively shoot more rays
  - Shadows
  - Reflections
  - Ambient occlusion
- Most common
Normal shader
struct PerRayData_radiance
{
    float3 result;
};

rtDeclareVariable(PerRayData_radiance, prd_radiance, rtPayload,);
rtDeclareVariable(float3, shading_normal, attribute shading_normal,);

RT_PROGRAM void closest_hit_radiance()
{
    float3 worldnormal = normalize(rtTransformNormal(RT_OBJECT_TO_WORLD, shading_normal));
    prd_radiance.result = worldnormal * 0.5f + 0.5f;
}
Normal shader
Lambertian shader
rtBuffer<BasicLight> lights;

rtDeclareVariable(optix::Ray, ray, rtIncomingRay, );
rtDeclareVariable(float, t_hit, rtIntersectionDistance, );

RT_PROGRAM void closest_hit_radiance()
{
    float3 world_geo_normal = normalize( rtTransformNormal( RT_OBJECT_TO_WORLD, geometric_normal ) );
    float3 world_shade_normal = normalize( rtTransformNormal( RT_OBJECT_TO_WORLD, shading_normal ) );
    float3 ffnormal = faceforward( world_shade_normal, -ray.direction, world_geo_normal );
    float3 color = Ka * ambient_light_color;

    float3 hit_point = ray.origin + t_hit * ray.direction;

    for(int i = 0; i < lights.size(); ++i) { // Loop over lights
        BasicLight& light = lights[i];
        float3 L = normalize(light.pos - hit_point);
        float nDl = dot( ffnormal, L);

        if( nDl > 0 )
            color += Kd * nDl * light.color;
    }
    prd_radiance.result = color;
}
Lambertian shader
Adding shadows
Ray Payloads

- Can define arbitrary data with the ray
- Sometimes called the “per ray data”
- Data can be passed down or up the ray tree (or both)
- Just a user-defined struct accessed by all shader programs
- Varies per ray type
for(int i = 0; i < lights.size(); ++i) {
    BasicLight light = lights[i];
    float3 L = normalize(light.pos - hit_point);
    float nDl = dot(ffnormal, L);

    if( nDl > 0.0f ){
        // cast shadow ray
        PerRayData_shadow shadow_prd;
        shadow_prd.attenuation = 1.0f;
        float Ldist = length(light.pos - hit_point);
        Ray shadow_ray = make_Ray(hit_point, L, 1, scene_epsilon, Ldist);
        rtTrace(top_shadower, shadow_ray, shadow_prd);
        float light_attenuation = shadow_prd.attenuation;

        if( light_attenuation > 0.0f ){
            float3 Lc = light.color * light_attenuation;
            color += Kd * nDl * Lc;
        }
    }
}
Adding shadows
Adding reflections
struct PerRayData_radiance {
    float3 result;
    float  importance;
    int    depth;
};

... 

float importance = prd.importance * luminance( reflectivity );

// reflection ray
if ( importance > importance_cutoff && prd.depth < max_depth ) {
    PerRayData_radiance refl_prd;
    refl_prd.importance = importance;
    refl_prd.depth = prd.depth + 1;
    float3 R = reflect( ray.direction, ffnormal );
    Ray refl_ray = make_ray( hit_point, R, 0,
                                scene_epsilon, RT_DEFAULT_MAX );
    rtTrace( top_object, refl_ray, refl_prd);
    color += reflectivity * refl_prd.result;
}
Adding reflections
Environment map
Miss program

- Defines what happens when a ray misses all objects
- Accesses ray payload
- Usually – background color
rtDeclareVariable(float3, bg_color, ,);
rtDeclareVariable(PerRayData_Radiance, prd_radiance, ,);
RT_PROGRAM void miss()
{
   prd_radiance.result = bg_color;
}
rtTextureSampler<float4, 2> envmap;
rtDeclareVariable(PerRayData_Radiance, prd_radiance, ,);

RT_PROGRAM void envmap_miss()
{
    float theta = atan2f(ray.direction.x, ray.direction.z);
    float phi = M_PIf * 0.5f - acosf(ray.direction.y);
    float u = (theta + M_PIf) * (0.5f * M_1_PIf);
    float v = 0.5f * (1.0f + sin(phi));
    prd_radiance.result = make_float3(tex2D(envmap, u, v));
}
Environment map
Schlick approximation
float3 r = fresnel_schlick(-dot(ffnormal, ray.direction), reflectivity_n);
float importance = prd.importance * luminance(r);

// reflection ray
if( importance > importance_cutoff && prd.depth < max_depth) {
    PerRayData_radiance refl_prd;
    refl_prd.importance = importance;
    refl_prd.depth = prd.depth + 1;
    float3 R = reflect(ray.direction, ffnormal);
    Ray refl_ray = make_ray(hit_point, R, 0,
        scene_epsilon, RT_DEFAULT_MAX);
    rtTrace(top_object, refl_ray, refl_prd);
    color += r * refl_prd.result;
}
Schlick approximation
Tiled floor
... float t_hit = incoming_ray_t.get();
float3 hit_point = ray.origin + t_hit * ray.direction;

float v0 = dot(tile_v0, hit_point);
float v1 = dot(tile_v1, hit_point);
v0 = v0 - floor(v0);
v1 = v1 - floor(v1);

float3 local_Kd;
if( v0 > crack_width && v1 > crack_width ){
    local_Kd = Kd;
} else {
    local_Kd = crack_color;
}
...
Tiled floor
Rusty metal
Direct port of Larry Gritz’ rusty metal shader
Rusty metal
Adding primitives
Intersection program

- Determines if/where ray hits an object
- Sets attributes (normal, texture coordinates)
  - Used by closest hit shader for shading
- Selects which material to use
- Used for
  - Programmable surfaces
  - Allowing arbitrary triangle buffer formats
- Etc.
Convex hull object

- Defined by a set of planes
- Created by the host
- Simple algorithm can handle any number of planes
  - Find last plane “entered”
  - Find first plane “exited”
  - Degenerate interval: miss
rtBuffer<float4> planes;
RT_PROGRAM void chull_intersect(int primIdx)
{
    const Ray ray = incoming_ray.get();

    int n = planes.size();
    float t0 = -FLT_MAX;
    float t1 = FLT_MAX;
    float3 t0_normal = make_float3(0);
    float3 t1_normal = make_float3(0);
    for(int i = 0; i < n; ++i ) {
        float4 plane = planes[i];
        float3 n = make_float3(plane);
        float d = plane.w;
        float denom = dot(n, ray.direction);
        float t = -(d + dot(n, ray.origin))/denom;
        if( denom < 0){
            // enter
            if(t > t0){
                t0 = t;
                t0_normal = n;
            }
        } else {
            //exit
            if(t < t1){
                t1 = t;
                t1_normal = n;
            }
        }
    }
    if(t0 > t1)
        return;
    if(rtPotentialIntersection( t0 )){
        shading_normal = geometric_normal = t0_normal;
        rtReportIntersection(0);
    } else if(rtPotentialIntersection( t1 )){
        shading_normal = geometric_normal = t1_normal;
        rtReportIntersection(0);
    }
}
Adding primitives
Tweaking the shadow
Any hit program

- Defines what happens when a ray attempts to hit an object
- Executed for all intersections along a ray
- Can optionally:
  - Stop the ray immediately (shadow rays)
  - Ignore the intersection and allow ray to continue (alpha transparency)
rtDeclareVariable(PerRayData_shadow, prd_shadow, ,);

RT_PROGRAM void any_hit_shadow()
{
    // this material is opaque,
    // so it fully attenuates all shadow rays
    prd_shadow.attenuation = 0;

    rtTerminateRay();
}
rtDeclareVariable(float, shadow_attenuation);
RT_PROGRAM void glass_any_hit_shadow()
{
    Ray ray = incoming_ray.get();
    float3 world_normal = normalize( rtTransformNormal( RT_OBJECT_TO_WORLD, shading_normal ));
    float nDi = fabs(dot(world_normal, ray.direction));
    prd.attenuation *= 1-fresnel_schlick(nDi, 5, 1-shadow_attenuation, 1);
    rtIgnoreIntersection();
}
Tweaking the shadow
Environment map camera
Ray generation program

- Starts the ray tracing process
- Used for:
  - Camera model
  - Output buffer writes
- Can trace multiple rays
- Or no rays
rtDeclareVariable(uint2, launchIndex, rtLaunchIndex,);

RT_PROGRAM void pinhole_camera()
{
    float2 d = make_float2(launchIndex) / 
        make_float2(output_buffer.size()) * 2.f - 1.f;
    float3 ray_origin = eye;
    float3 ray_direction = normalize(d.x*U + d.y*V + W);

    Ray ray = make_ray(ray_origin, ray_direction, radiance_ray_type, 
        scene_epsilon, RT_DEFAULT_MAX);

    PerRayData_radiance prd;
    prd.importance = 1.f;
    prd.depth = 0;

    rtTrace(top_object, ray, prd);

    output_buffer[index] = make_color( prd.result );
}
RT_PROGRAM void env_camera()
{
    float2 d = make_float2(launchIndex) / make_float2(output_buffer.size());
    d = d * make_float2(2.0f * M_PIf, M_PIf) + make_float2(M_PIf, 0);
    float3 angle = make_float3(cos(d.x) * sin(d.y), -cos(d.y), sin(d.x) * sin(d.y));

    float3 ray_origin = eye;
    float3 ray_direction = normalize(angle.x*normalize(U) + angle.y*normalize(V) + angle.z*normalize(W));

    Ray ray = make_ray(ray_origin, ray_direction, radiance_ray_type, scene_epsilon, RT_DEFAULT_MAX);

    PerRayData_radiance prd;
    prd.importance = 1.f;
    prd.depth = 0;

    rtTrace(top_object, ray, prd);

    output_buffer[index] = make_color( prd.result );
}
Environment map camera
Next steps

- Multiple rays per pixel (raygen program)
- Image-based lighting (closest hit program)
- Ambient occlusion (closest hit program)
- Path tracer (new shaders, raygen program)
- Interaction with host code
Other features

- Powerful object model
  - All objects green except one
  - Different light source list for a single object
- Can use double precision arithmetic
- OptiX node-graph
  - Programmable traversal
  - Instancing
  - Built-in acceleration structures
    - BVH, SBVH, LBVH, kd-tree
  - Supports dynamic scenes
- Multiple "entry points"
  - Adaptive AA
  - Photon pass, gather pass
- Interop with OpenGL (and D3D soon)
  - Textures, VBOs, etc.
IMAGE SPACE GATHERING
Glossy Reflection
Algorithm Pipeline

1. Rasterize Requests
2. Ray Trace Reflections
3. Blur and Composite
ISG
PCF (Screen Space)
OptiX SDK Release

Available for free “very soon” from http://developer.nvidia.com
Questions?

arobison@nvidia.com

http://www.nvidia.com