RAY TRACING WITH OPTIX
A TUTORIAL FOR DEVELOPERS
David McAllister, James Bigler, Brandon Lloyd
RAY TRACING IN THE ABSTRACT

- Given a ray \((O, D)\) and a geometric dataset find
  - any hit
  - closest hit
  - all hits

- Current datasets \(~1M \rightarrow 100M\) primitives, usually triangles
- Use a spatial data structure optimized for these operations
- Datasets can also include GB of other data like textures
OPTIX RAY TRACING TALKS AT GTC

- Nvidia Keynote - Tue. 9:00
- Pixar Keynote - Wed. 11:00
- S4260 - Game Physics and Graphics APIs for Scientific Computing - Tue. 16:00
- S4359 - EM Wave Propagation Using OptiX for Simulation of Car-to-Car-Comm - Wed. 14:00
- S4697 - Realtime Preview For VFX: Challenges and Rewards - Wed. 14:00
- S4690 - Implementing OptiX Ray Tracing Features into FurryBall GPU Renderer - Wed. 17:30
- S4597 - Advanced OptiX Programming - Thur. 9:00
- S4400 - Petascale Molecular Ray Tracing: Accelerating VMD/Tachyon with OptiX - Thur. 10:00
- S4312 - QUIC EnvSim: Radiative Heat Transfer in Vegetative and Urban Environments with NVIDIA OptiX, Thur. 14:30
RENDERING WITH RAY TRACING
RAY CASTING (APPEL, 1968)
RECURSIVE RAY TRACING (WHITTED, 1980)
DISTRIBUTION RAY TRACING (COOK, 1984)
PATH TRACING (KAJIYA, 1986)

Figure 6. A sample image. All objects are neutral grey. Color on the objects is due to caustics from the green glass balls and color bleeding from the base polygon.
WHY RAY TRACING?

- Ray tracing unifies rendering of visual phenomena
  - fewer algorithms with fewer interactions between algorithms
- Easier to combine advanced visual effects robustly
  - soft shadows
  - subsurface scattering
  - indirect illumination
  - transparency
  - reflective & glossy surfaces
  - depth of field
  - ...
REAL TIME PATH TRACING

- What would it take?
  - 4 rays / sample
  - 50 samples / pixel
  - 2M pixels / frame
  - 30 frames / second
  - 12B rays / second

- GeForce GTX Titan:
  - 400M rays / second
  - Need 30X speedup
RAY TRACING REGIMES

- Interactive
- Batch

Real-time

today
HOW TO OPTIMIZE RAY TRACING

- Better hardware (GPUs)
- Better software (Algorithmic improvement)
- Better middleware (Tune for the architecture)
GPUS - THE PROCESSOR FOR RAY TRACING

- Abundant parallelism, massive computational power
- GPUs excel at shading
- Opportunity for hybrid algorithms
ACCELERATION STRUCTURES
OPTIX PRIME

- Specialized for ray tracing (no shading)
- Replaces rtuTraversal (rtuTraversal is still supported)
- Improved performance
  - Uses latest algorithms from NVIDIA Research
    - ray tracing kernels [Aila and Laine 2009; Aila et al. 2012]
    - Treelet Reordering BVH (TRBVH) [Karras 2013]
  - Can use CUDA buffers as input/output
  - Support for asynchronous computation
- Designed with an eye towards future features
API OVERVIEW

- C API with C++ wrappers
- API Objects
  - Context
  - Buffer Descriptor
  - Model
  - Query
CONTEXT

- Context tracks other API objects and encapsulates the ray tracing backend
- Creating a context
  ```c
  RTPResult
  rtpContextCreate(RTPcontextType type, RTPcontext* context)
  ```
- Context types
  ```c
  RTP_CONTEXT_TYPE_CPU
  RTP_CONTEXT_TYPE_CUDA
  ```
- Default for CUDA backend uses all available GPUs
  - Selects “Primary GPU” and makes it the current device
  - Primary GPU builds acceleration structure
Selecting devices:

```c
rtpContextSetCudaDeviceNumbers(
    RTPcontext context,
    int deviceCount,
    const int* deviceNumbers )
```

- First device is used as the primary GPU

Destroying the context

- destroys objects created by the context
- synchronizes the CPU and GPU
BUFFERS DESCRIPTOR

- Buffers are allocated by the application
- Buffer descriptors encapsulate information about the buffers

```c
rtpBufferDescCreate(
    RTPcontext    context,
    RTPbufferformat format,
    RTPbuffertype    type,
    void*     buffer,
    RTPbufferdesc* desc )
```

- Specify region of buffer to use (in elements)

```c
rtpBufferDescSetRange( RTPbufferdesc desc, int begin, int end )
```
Variable stride supported for vertex format

\texttt{rtpBufferDescSetStride}

- Allows for vertex attributes
• Formats
  
  RTP_BUFFER_FORMAT_INDICES_INT3
  RTP_BUFFER_FORMAT_VERTEX_FLOAT3,
  RTP_BUFFER_FORMAT_RAY_ORIGIN_DIRECTION,
  RTP_BUFFER_FORMAT_RAY_ORIGIN_TMIN_DIRECTION_TMAX,
  RTP_BUFFER_FORMAT_HIT_T_TRIID_U_V
  RTP_BUFFER_FORMAT_HIT_T_TRIID
  ...

• Types

  RTP_BUFFER_TYPE_HOST
  RTP_BUFFER_TYPE_CUDA_LINEAR
A model is a set of triangles combined with an acceleration data structure:
- `rtpModelCreate`
- `rtpModelSetTriangles`
- `rtpModelUpdate`

Asynchronous update:
- `rtpModelFinish`
- `rtpModelGetFinished`
**QUERY**

- Queries perform the ray tracing on a model
  - `rtpQueryCreate`
  - `rtpQuerySetRays`
  - `rtpQuerySetHits`
  - `rtpQueryExecute`

- **Query types**
  - `RTP_QUERY_TYPE_ANY`
  - `RTP_QUERY_TYPE_CLOSEST`

- **Asynchronous query**
  - `rtpQueryFinish`
  - `rtpQueryGetFinished`
BUILD PERFORMANCE

Speedup over SBVH

Arabic Armadillo Babylonian Bar Blade Bubs Buddha City Conference Dragon Fairy Hairball Italian Motor Mustang PowerPlant Sibenik Soda Sponza Veyron
RAY TRACING PERFORMANCE

The chart shows the speedup over `rtuTraversal` for various benchmarks. The benchmarks include Arabic, Armadillo, Babylonian, Bar, Blade, Bubs, Buddha, City, Conference, Dragon, Fairy, Hairball, Italian, Motor, Mustang, PowerPlant, Sibenik, Soda, Sponza, and Veyron. The `rtuTraversal` performance is normalized to a value of 1.0, and the speedup is measured as a ratio relative to this baseline.
FUTURE

- Features we want to implement
  - Animation support (refit/refine)
  - Instancing
  - Large-model optimizations
ART AND ANIMATION STUDIOS

- Jan Tománek, CEO
- FurryBall 4.6
OPTIX GOALS

- High performance ray tracing
- Simpler ray tracing
- Hide GPU-specific details
- Express most ray tracing algorithms
- Leverage CUDA architecture and compiler infrastructure
GENERAL PURPOSE RAY TRACING

- 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
- Algorithm agnostic
  - User defined ray data
  - Programmable intersection
  - Interoperate with rasterization pipeline
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
OTHER OPTIX FEATURES

- OptiX node graph
  - Programmable traversal
  - Instancing
  - Dynamic scenes

- Double precision arithmetic

- Interop with CUDA, OpenGL and D3D
  - Textures, VBOs, etc.
  - Hybrid rasterization and ray tracing
OptiX 3.5

The NVIDIA® OptiX™ Ray Tracing Engine is a programmable ray tracing framework helping software developers build ray tracing applications in a fraction of the time of conventional methods, that then run exceedingly fast on NVIDIA GPUs. Unlike a renderer with a prescribed look, a language limited to rendering, or prescribed solutions with fixed data structures, the OptiX engine is extremely general - enabling developers to quickly accelerate whatever ray tracing task they wish, integrate it as needed, and run it on commonly available hardware.

Rapid Ray Tracing Development

The OptiX engine takes care of the "heavy lifting" associated with ray tracing, giving developers more time to concentrate on technique with relatively small programs that leverage the latest GPU advances. A single ray execution model makes building custom techniques straightforward, while state of the art acceleration structures (such as BVH and KD trees), cutting-edge traversal algorithms, load balancing, recursion, parallelism (across CUDA Cores and GPUs), out of core processing, and interop with OpenGL, Direct3D and CUDA, makes ray tracing development far easier than alternative approaches.

Ray Tracing Flexibility

OptiX easily extends beyond image creation by enabling rays to gather and carry custom payloads. The data fed to OptiX is also programmable, enabling custom shading techniques, programmable intersection for procedural definitions, and programmable cameras for customized ray dispatching. This flexibility enables OptiX to accelerate ray traced rendering algorithms ranging from the highly interactive to the ultra-realistic, while also accommodating disciplines such as acoustics, ballistics, collision analysis, radiation reflectance, or volume calculations - wherever intensive ray tracing calculations are employed.

Rapid Evolution

Applications employing OptiX continually increase in speed by using consistent APIs that exploit the latest advances from both new hardware and ray tracing research at NVIDIA. As a compiler based technology, OptiX builds the optimal runtime for the processors it finds, freeing developers from having to optimize their applications per GPU architecture.

Additional Information

To see the latest release notes on OptiX, click here.

The OptiX QuickStart Guide walks through several tutorials to get you started with OptiX.

Learn more about using OptiX with the Programming Guide.

To read the details on using the OptiX API, click here.
HOW OPTIX LINKS YOUR CODE

- Ray Generation
- Material Shading
- Object Intersection
- JIT Compiler
- Acceleration Structures
- Scheduling

CUDA C shaders from user programs

OptiX API
OVERVIEW - API OBJECTS

- Context
  - Program
  - Variable
  - GeometryInstance
  - Geometry
  - Material
  - Acceleration
  - Buffer
  - TextureSampler
- Group
- GeometryGroup
- Transform
- Selector
API OBJECTS - CONTEXT

- Manages API Object State
  - Program Loading
  - Validation and Compilation

- Manages Acceleration Structures
  - Building and Updating

- Provides Entry Points into the system
  - rtContextLaunch*D()
ENTRY POINTS AND RAY TYPES

Context

Entry Point 1
- Ray Generation 1
- Exception 1

Entry Point 2
- Ray Generation 2
- Exception 2

Trace
ENTRY POINTS AND RAY TYPES

Trace

Ray Shading

Material Programs

Material
Ray Type

0
Closest Hit
Any Hit

1
Closest Hit
Any Hit

2
Closest Hit
Any Hit

3
Closest Hit
Any Hit
API OBJECTS - NODES

- Nodes contain children
  - Other nodes
  - Geometry Instances

- Transforms hold matrices
  - Applied to all children

- Selectors have Visit programs
  - Provide programmable selection of children
  - Similar to “switch nodes”
  - Can implement LOD systems

- Acceleration Structures
  - Builds over children of attached node
THE OBJECT HIERARCHY

- Context
  - Group
    - GeometryGroup
      - GeometryInstance
      - GeometryInstance
      - GeometryInstance
    - GeometryGroup
      - GeometryInstance
**API OBJECTS - GEOMETRY**

- **GeometryInstance** binds:
  - Geometry object
  - A collection of Materials
    - Indexed by argument from intersection

- **Geometry**
  - A collection of primitives
    - Intersection Program
    - Bounding Box Program

- **Material**
  - Any Hit Program
  - Closest Hit Program
DEFORMABLE OBJECTS

1. Primitives
   Deform

2. Groups and Acceleration Marked Dirty

3. Context updates Acceleration Structures
API OBJECTS - DATA MANAGEMENT

- Supports 1D, 2D and 3D buffers
- Buffer formats
  - RT_FORMAT_FLOAT3
  - RT_FORMAT_UNSIGNED_BYTE4
  - RT_FORMAT_USER, etc.
- Other API Interoperability
  - e.g. create buffers from CUDA, OpenGL or D3D buffer objects
- TextureSamplers reference Buffers
  - Attach buffers to MIP levels, array slices, etc.
API OBJECTS - PROGRAMMABILITY

- Runs on CUDA
  - Cg-like vectors plus pointers
  - Uses PTX, CUDA's virtual assembly language
  - C++ wrapper for use with NVCC compiler

- Implements recursion and dynamic dispatch
  - Intrinsic functions: rtTrace(), rtReportIntersection(), etc.

- Programs reference variables by name

- Variables are defined by
  - Static initializers
  - Binding to API Objects in the hierarchy
VARIABLES

- Variables are one of:
  - A small primitive type (float4, matrix, ...)
  - A small user defined type
  - A handle to a buffer (1D, 2D, 3D)
  - A texture
  - A handle to a callable program
  - A buffer ID
  - A program ID
VARIABLE SCOPING

Context

GeometryInstance

Closest Hit Program

Material

Program

Definition: Color = blue

Definition: Color = red

Reference: Color

Definition: Color = blue

GeometryInstance

Closest Hit Program

Material
BUFFER IDS (V3.5)

- Previously only attachable to Variables
- With a Buffer API object, request the ID (rtBufferGetId)
- Use ID
  - In a buffer
    - rtBuffer<rtBufferId<float3,1>, 1> buffers;
    - float3 val = buffers[i][j];
  - Passed as arguments*
    - float work(rtBufferId<float3,1> data);
  - Stored in structs*
    - struct MyData { rtBufferId<float3,1>; int stuff; };

* Can thwart some optimizations
CALLABLE PROGRAM IDS (V3.6)

- Think of them as a functor (function pointer with data)
  - PTX (RTprogram)
  - Variables attached to RTprogram API object
- With a RTprogram API object, request the ID (`rtProgramGetId`)
- Use ID
  - In a buffer
    - `rtBuffer<rtCallableProgramId<int,int>, 1> programs;`
    - `int val = programs[i](4);`
  - As a variable
    - `typedef rtCallableProgramId<int,int> program_t;`
    - `rtDeclareVariable(program_t, program,,);`
    - `int val = program(3);`
- Passed as arguments*
- Stored in structs*

* Can thwart some optimizations
C HOST API SAMPLE

RTresult RTAPI rtContextCreate (RTcontext* context);
RTresult RTAPI rtContextDestroy (RTcontext context);
RTresult RTAPI rtContextDeclareVariable (RTcontext context, const char* name, RTvariable* v);
RTresult RTAPI rtContextSetRayGenerationProgram(RTcontext context, unsigned int entry_point_index, RTprogram program);
RTresult RTAPI rtBufferCreate (RTcontext context, unsigned int bufferdesc, RTbuffer* buffer);
RTresult RTAPI rtBufferSetFormat (RTbuffer buffer, RTformat format);
RTresult RTAPI rtBufferMap (RTbuffer buffer, void** user_pointer);
RTresult RTAPI rtBufferUnmap(RTbuffer buffer);
RTresult RTAPI rtProgramCreateFromPTXString(RTcontext context, const char* ptx, const char* program_name, RTprogram* program);
RTresult RTAPI rtProgramCreateFromPTXFile (RTcontext context, const char* filename, const char* program_name, RTprogram* program);
RTresult RTAPI rtContextLaunch2D(RTcontext context, unsigned int entry_point_index, RTsize image_width, RTsize image_height);
Context* context = Context::create();
context["max_depth"]->setInt( 5 );
context["scene_epsilon"]->setFloat( 1.e-4f );
// Ray gen program
Program ray_gen_program = context->createProgramFromPTXFile( "myprogram.ptx","pinhole_camera"
context->setRayGenerationProgram( 0, ray_gen_program );

BasicLight lights[] = { ..... };
Buffer light_buffer = context->createBuffer(RT_BUFFER_INPUT);
light_buffer->setFormat(RT_FORMAT_USER);
light_buffer->setElementSize(sizeof(BasicLight));
light_buffer->setSize( sizeof(lights)/sizeof(lights[0]) );
memcpy(light_buffer->map(), lights, sizeof(lights));
light_buffer->unmap();
context["lights"]->set(light_buffer);
LIFE OF A RAY

1. Ray Generation
2. Intersection
3. Shading

Pinhole Camera
Payload float3 color
Ray-Sphere Intersection
Lambertian Shading
OPTIX EXECUTION MODEL

Launch
- rtContextLaunch
- Ray Generation Program
- Exception Program

Traverse
- Node Graph Traversal
- Selector Visit Program
- Acceleration Traversal
- Intersection Program

Shade
- Miss Program
- Closest Hit Program
- Any Hit Program
RT_PROGRAM void pinhole_camera()
{
    float2 d = make_float2(launch_index) / 
               make_float2(launch_dim) * 2.f - 1.f;
    float3 ray_origin = eye;
    float3 ray_direction = normalize(d.x*U + d.y*V + W);
    optix::Ray ray = optix::make_Ray(ray_origin, ray_direction,
                                     radiance_ray_type, scene_epsilon, RT_DEFAULT_MAX);

    PerRayData_radiance prd;
    rtTrace(top_object, ray, prd);
    output_buffer[launch_index] = make_color(prd.result);
}

- Interconnection of programs defines the outcome
- Data associated with ray is programmable
- Input “language” is CUDA C/C++
  - No new language to learn
  - Powerful language features available immediately
  - Can also take raw PTX as input
- Caveat: still need to use it responsibly to get performance
SPLIT SHADING MODEL

- 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
FLEXIBLE INTERSECTION

- Intersection (miss)
- Intersection (hit)
- Any Hit
- Any Hit: rtlgnoreIntersection
- Closest Hit
FLEXIBLE INTERSECTION

Intersection (hit)
Any Hit: rtIgnoreIntersection

Intersection (hit)
Any Hit: rtTerminateRay

Closest Hit
PER RAY DATA AND ATTRIBUTES

Per Ray Data
- User-defined struct attached to rays
- Can be used to pass data up and down the ray tree
- Varies per Ray Type

Arbitrary Attributes
- Produced by Intersection Programs
- Consumed by Any Hit and Closest Hit Programs
CLOSEST HIT PROGRAM ("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
MOVING PICTURE COMPANY

- Damien Fagnau
  - Global Head, VFX Operations
OPTIX 3.5 WHAT’S NEW

- **OptiX Prime**
  for blazingly fast traversal & intersection ($\pm 300m$ rays/sec/GPU)
  - You give the triangles and rays, you get the intersections, in 5 lines of code

- **TRBVH Builder**
  builds +100X faster, runs about as fast as SBVH (previous fastest)
  - Part of OptiX Prime, also in OptiX core
  - Does require more memory (to be improved later this year)

- **GK110B Optimizations** (K40, K6000, GTX Titan Black)
  +25% more performance

- **Bindless Buffers & Buffers of Buffers**
  - More flexibility with callable programs (e.g., shade trees)
rtContextCompile()

3-7X faster

- Still, you should avoid recompiles if possible.
UPDATED SUPPORT

- Visual Studio 2012 support
- CUDA 5.5 support
VISUAL STUDIO OPTIX WIZARD

Welcome to the NVIDIA OptiX Wizard

To create a new project which uses the OptiX engine, please provide the following paths on your system:

- OptiX SDK installation path: [path]
- CUDA Toolkit installation path: [path]

Generate OptiX Project
VERTEX LIGHT BAKING

- Working with Bungie
  - Available publicly. *Just ask us.*

- Compared to textures...
  - Less memory & bandwidth
  - No $u,v$ parameterization
  - Good for low-frequency effects

- Over 250,000 baking jobs done
OPTIX 3.5 SDK

- Available for free:
- Windows, Linux, Mac
HIGH PERFORMANCE GRAPHICS 2014

- Lyon, France
- June 23-25
- Paper Submissions Due: April 4
- Poster Submissions Due: May 16
- Hot3D Submissions Due: May 23

www.highperformancegraphics.org