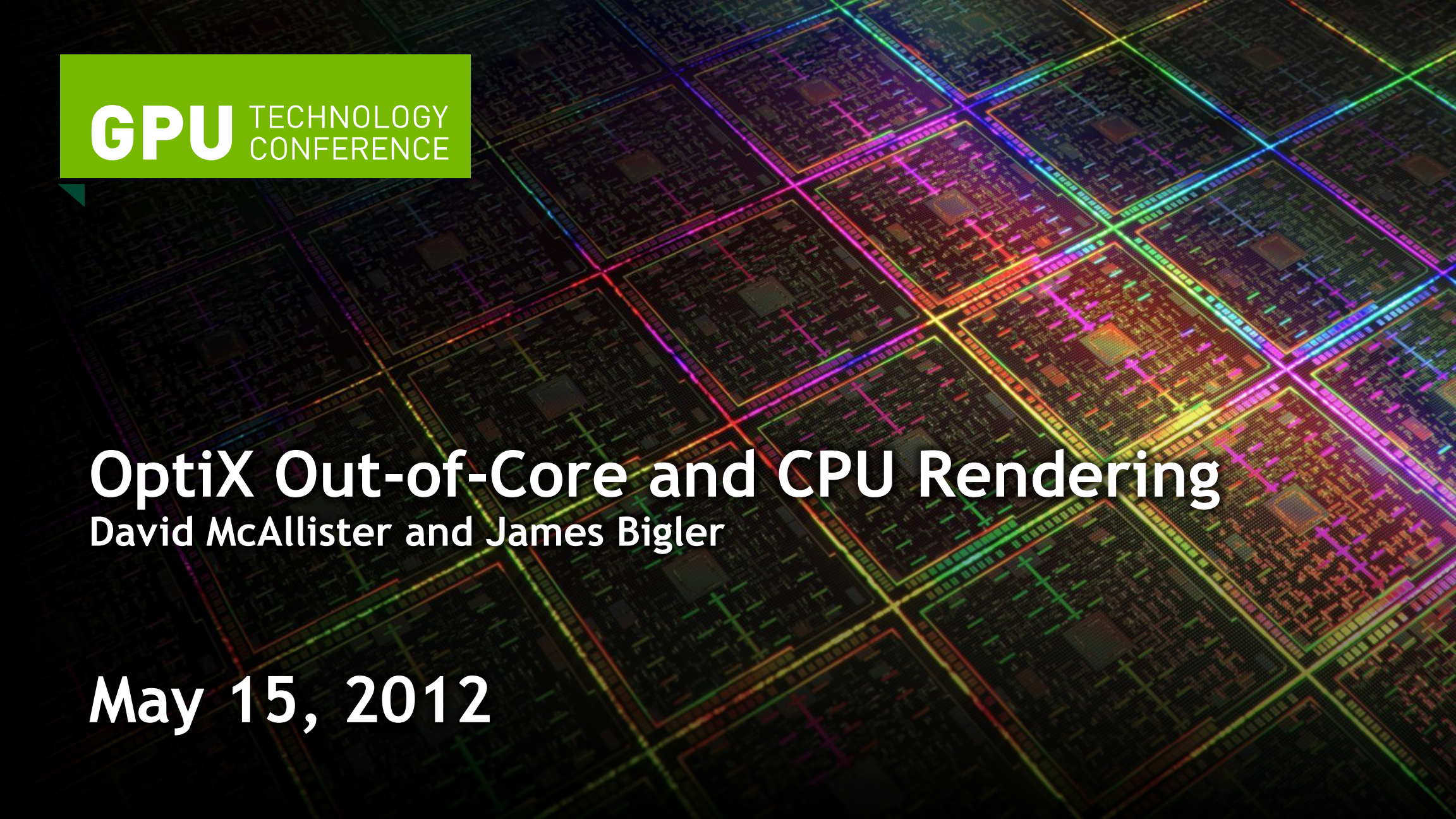
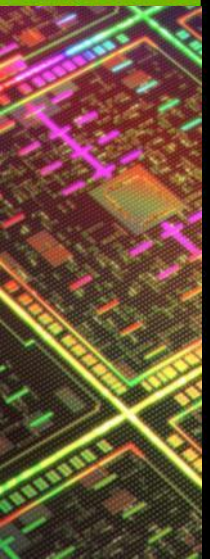


OptiX Out-of-Core and CPU Rendering

David McAllister and James Bigler

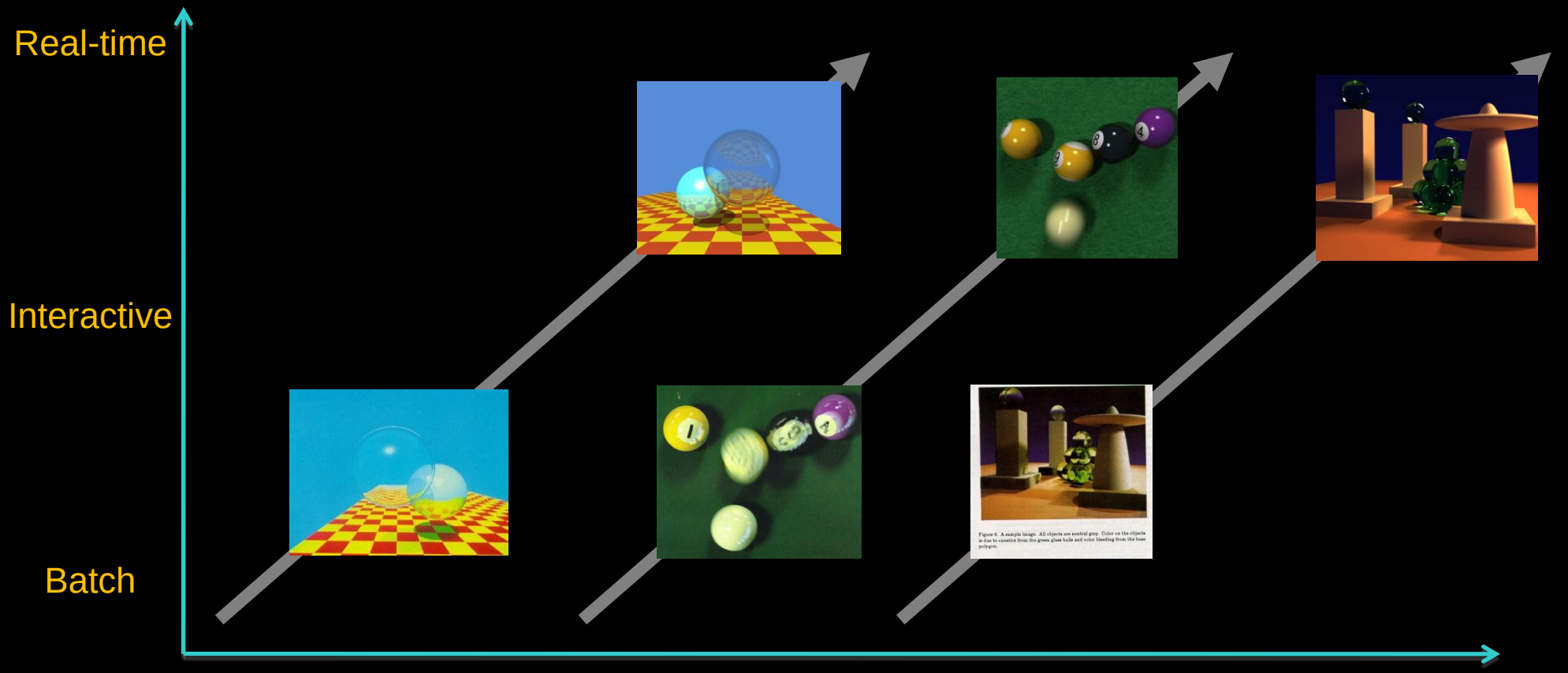
May 15, 2012



Agenda

- Ray Tracing Complexity
- OptiX Intro
- OptiX Internals
- CPU Fallback
- GPU Paging

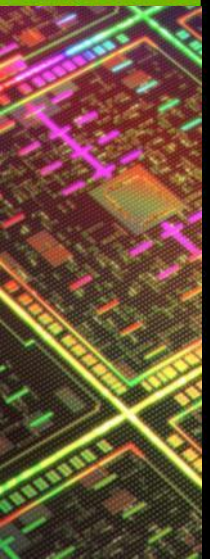
Ray Tracing Regimes



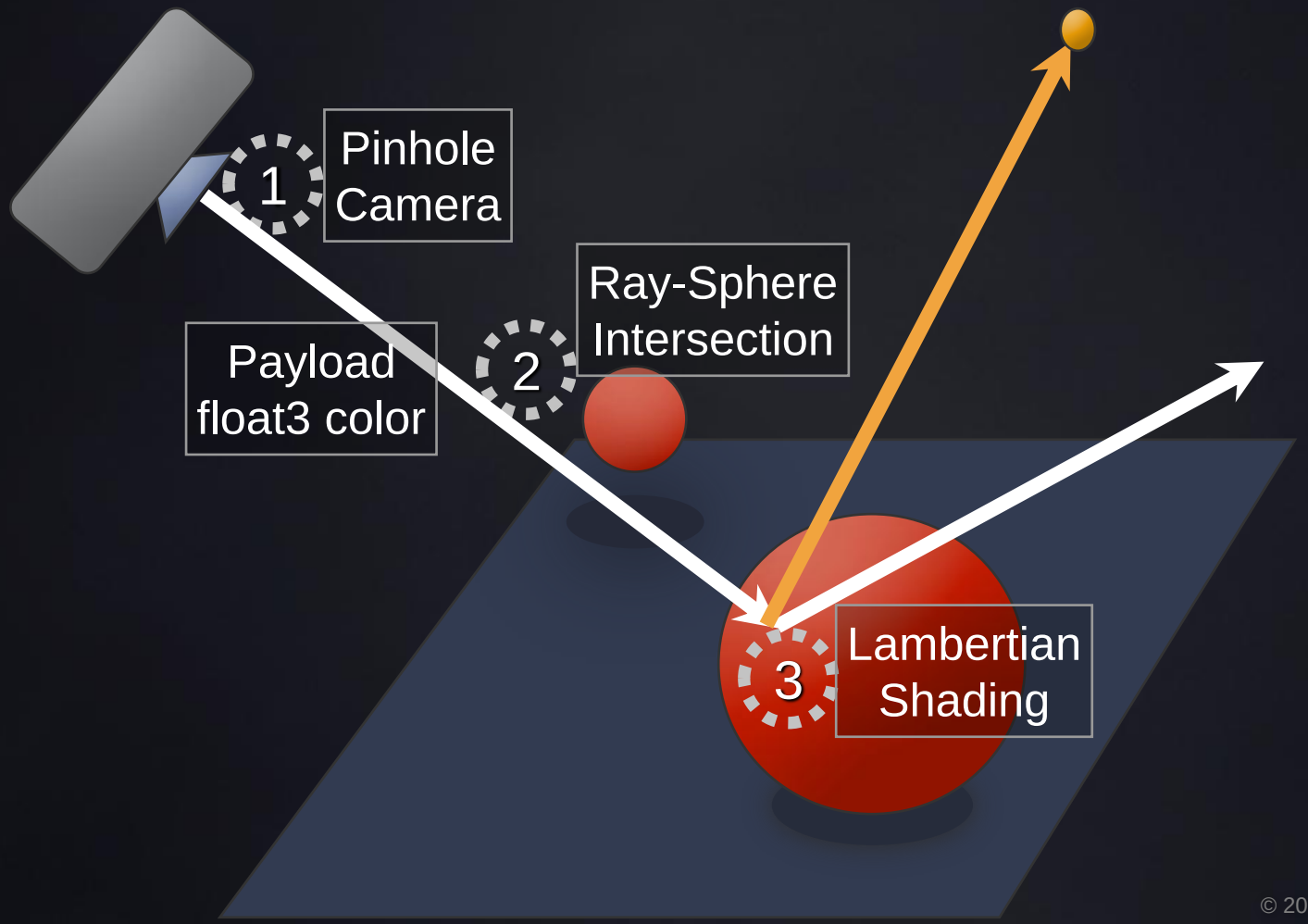
How to optimize ray tracing (or anything)

1. GPUs
2. Algorithmic improvement
3. Tune for the architecture

1. Better hardware
2. Better software
3. Better middleware



Life of a ray



1 Ray Generation

2 Intersection

3 Shading

Life of a ray

1 Pinhole Camera

```
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 );
}
```

2 Ray-Sphere Intersection

```
RT_PROGRAM void intersect_sphere()
{
  float3 O = ray.origin - center;
  float3 D = ray.direction;
  float b = dot(O, D);
  float c = dot(O, O)-radius*radius;
  float disc = b*b-c;
  if(disc > 0.0f){
    float sdisc = sqrtf(disc);
    float root1 = (-b - sdisc);
    bool check_second = true;
    if( rtPotentialIntersection( root1 ) ) {
      shading_normal = geometric_normal = (O + root1*D)/radius;
      if(rtReportIntersection(0))
        check_second = false;
    }
    if(check_second) {
      float root2 = (-b + sdisc);
      if( rtPotentialIntersection( root2 ) ) {
        shading_normal = geometric_normal = (O + root2*D)/radius;
        rtReportIntersection(0);
      }
    }
  }
```

3 Lambertian Shading

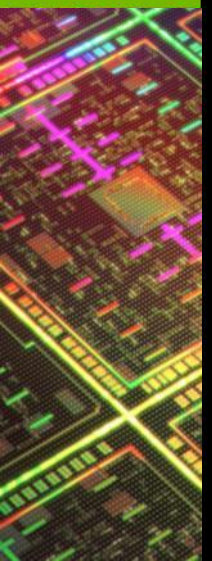
```
RT_PROGRAM void closest_hit_radiance3()
{
  float3 world_geo_normal   = normalize( rtTransformNormal( RT_OBJECT_TO_WORLD, geometric_normal ) );
  float3 world_shade_normal = normalize( rtTransformNormal( RT_OBJECT_TO_WORLD, shading_normal ) );
  float3 ffnormal     = faceforward( world_shade_normal, -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) {
    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 = make_float3(1.0f);
      float Ldist = length(light.pos - hit_point);
      optix::Ray shadow_ray( hit_point, L, shadow_ray_type, scene_epsilon, Ldist );
      rtTrace(top_shadower, shadow_ray, shadow_prd);
      float3 light_attenuation = shadow_prd.attenuation;

      if( fmaxf(light_attenuation) > 0.0f ){
        float3 Lc = light.color * light_attenuation;
        color += Kd * nDl * Lc;

        float3 H = normalize(L - ray.direction);
        float nDh = dot( ffnormal, H );
        if(nDh > 0)
          color += Ks * Lc * pow(nDh, phong_exp);
      }

    }
  }
  prd_radiance.result = color;
}
```

Program objects (shaders)

```
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 );
}
```

- Input “language” is based on CUDA C/C++
  - No new language to learn
  - Powerful language features available immediately
  - Can also take raw PTX as input
- Data associated with ray is programmable
- Caveat: still need to use it responsibly to get performance

How OptiX Links Your Code



Ray Generation

Material Shading

Object Intersection

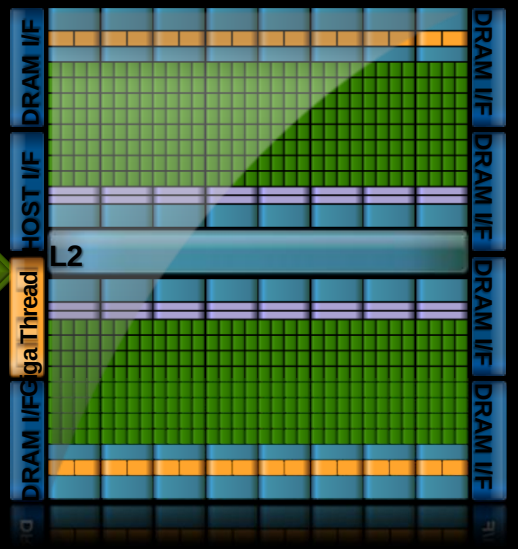
CUDA C shaders from user programs

Acceleration Structures

JIT Compiler

Scheduling

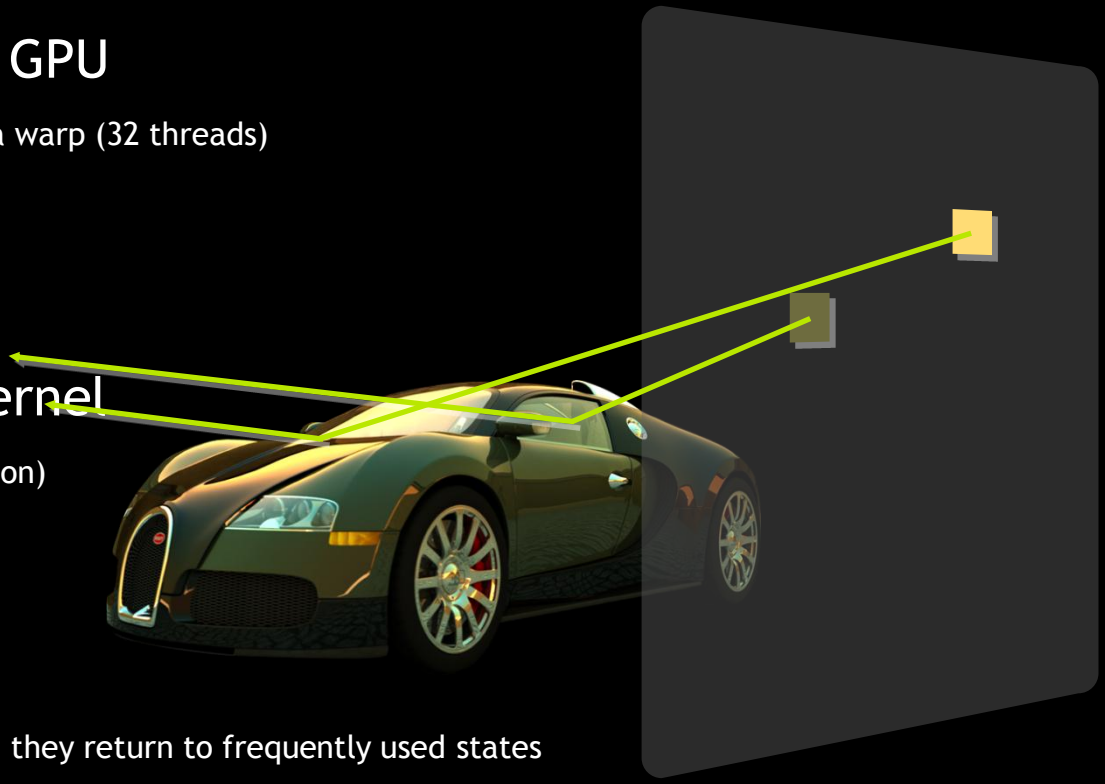
OptiX API



GPU Execution via CUDA

Compilation

- Goal: execute efficiently on a GPU
  - Must manage execution coherence within a warp (32 threads)
  - Must manage data coherence
  - Minimize local state
  - Minimize context switch overhead
- Strategy: compile to a megakernel
  - All programs (traversal, shading, intersection)
  - Utilize dynamic load-balancing
  - Some divergence unavoidable
- Key observation
  - Although threads may temporarily diverge, they return to frequently used states



Compilation to state machine

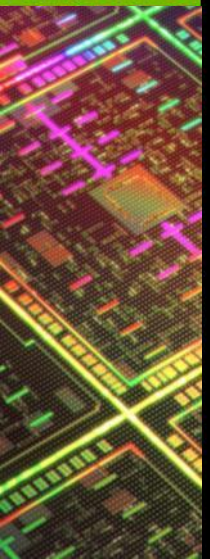
```
RT_PROGRAM void pinhole_camera()
{

  Ray ray = make_ray(…);

  PerRayData_radiance prd;

  rtTrace(top_object, ray, prd);

  output_buffer[index] =
       make_color( prd.result );
}
```

```
RT_PROGRAM void pinhole_camera()
{

  Ray ray = make_ray(…);

  PerRayData_radiance prd;
  save prd, index;
  rtTrace(top_object, ray, prd);
  restore prd, index;
  output_buffer[index] =
       make_color( prd.result );
}
```

State 1

State 2

- Optix Just-in Time Compiler
- Inserts continuations
- Transforms to state machine
- Rewrites variable load/store for object model
- Inlines intrinsic functions

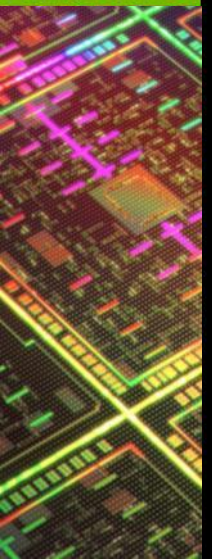


Step 1: Compile to PTX

```
for( int i = 0; i < 5; ++i ) {
  Ray ray = make_Ray(
  make_float3( i, 0, 0 ),
  make_float3( 0, 0, 1 ),
  0,  1e-4f, 1e20f );

  UserPayloadStruct payload;
  rtTrace(top_object, ray, payload );
}
```

nvcc

```
  ld.global.u32   %node, [top_object+0];
  mov.s32         %i, 0;
loop:
  call _rt_trace, ( %node, %i, 0, 0, 0, 0, 1,
                    0, 1e-4f, 1e20f, payload
  );
  add.s32         %i, %i, 1;
  mov.u32         %iend, 5;
  setp.ne.s32     %predicate, %i, %iend;
  @%predicate bra loop;
```

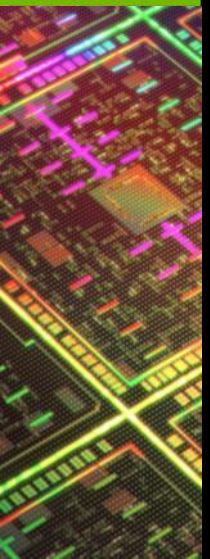


Step 2: Insert continuations

```
  ld.global.u32   %node, [top_object+0];
  mov.s32         %i, 0;
loop:
  call _rt_trace, ( %node, %i, 0, 0, 0, 0, 1,
                    0, 1e-4f, 1e20f, payload);
  add.s32         %i, %i, 1;
  mov.u32         %iend, 5;
  setp.ne.s32     %predicate, %i, %iend;
  @%predicate bra loop;
```

OptiX

```
  ld.global.u32   %node, [top_object+0];
  mov.s32         %i, 0;
loop:
  mov payload, %stack;
  save %i, %iend, %node;
  call _rt_trace, ( %node, %i, 0, 0, 0, 0, 1,
                    0, 1e-4f, 1e20f,payload);
  restore %i, %iend, %node;
  add.s32         %i, %i, 1;
  mov.u32         %iend, 5;
  setp.ne.s32     %predicate, %i, %iend;
  @%predicate bra loop;
```

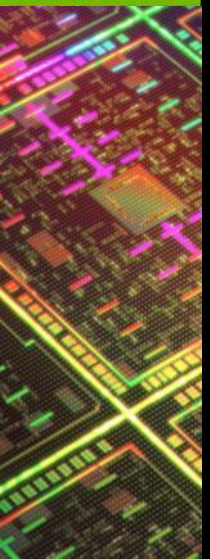


Step 3: Apply optimizations

```
  ld.global.u32   %node, [top_object+0];
  mov.s32         %i, 0;
loop:
  mov payload, %stack;
  save %i, %iend, %node;
  call _rt_trace, ( %node, %i, 0, 0, 0, 0, 1,
                    0, 1e-4f, 1e20f,payload);
  restore %i, %iend, %node;
  add.s32         %i, %i, 1;
  mov.u32         %iend, 5;
  setp.ne.s32     %predicate, %i, %iend;
  @%predicate bra loop;
```

OptiX

```
  ld.const.u32   %node, [top_object+0];
  mov.s32         %i, 0;
loop:
  mov payload, %stack;
  save %i;
  call _rt_trace, ( %node, %i, 0, 0, 0, 0,1,
                     0, 1e-4f, 1e20f,payload);
  restore %i;
  rematerialize %iend, %node;
  add.s32         %i, %i, 1;
  mov.u32         %iend, 5;
  setp.ne.s32     %predicate, %i, %iend;
  @%predicate bra loop;
```

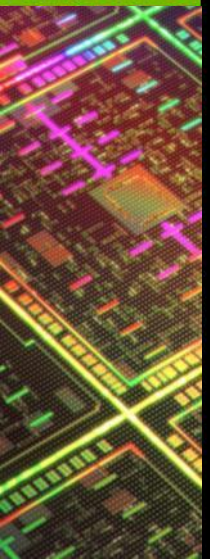


Step 4: Transform to state machine

```
  ld.const.u32    %node, [top_object+0];
  mov.s32          %i, 0;
loop:
  mov payload, %stack;
  save %i;
  call _rt_trace, ( %node, %i, 0, 0, 0, 0, 1,
                    0, 1e-4f, 1e20f,payload);
  restore %i;
  rematerialize %iend, %node;
  add.s32          %i, %i, 1;
  mov.u32          %iend, 5;
  setp.ne.s32      %predicate, %i, %iend;
  @%predicate bra loop;
```

OptiX

```
state 1:
  ld.const.u32    %node, [top_object+0];
  mov.s32          %i, 0;
loop:
  mov payload, %stack;
  save %i;
  bra mainloop;

state 2:
  restore %i;
  rematerialize %iend, %node;
  add.s32          %i, %i, 1;
  mov.u32          %iend, 5;
  setp.ne.s32      %predicate, %i, %iend;
  @%predicate bra loop;
  bra mainloop;
```



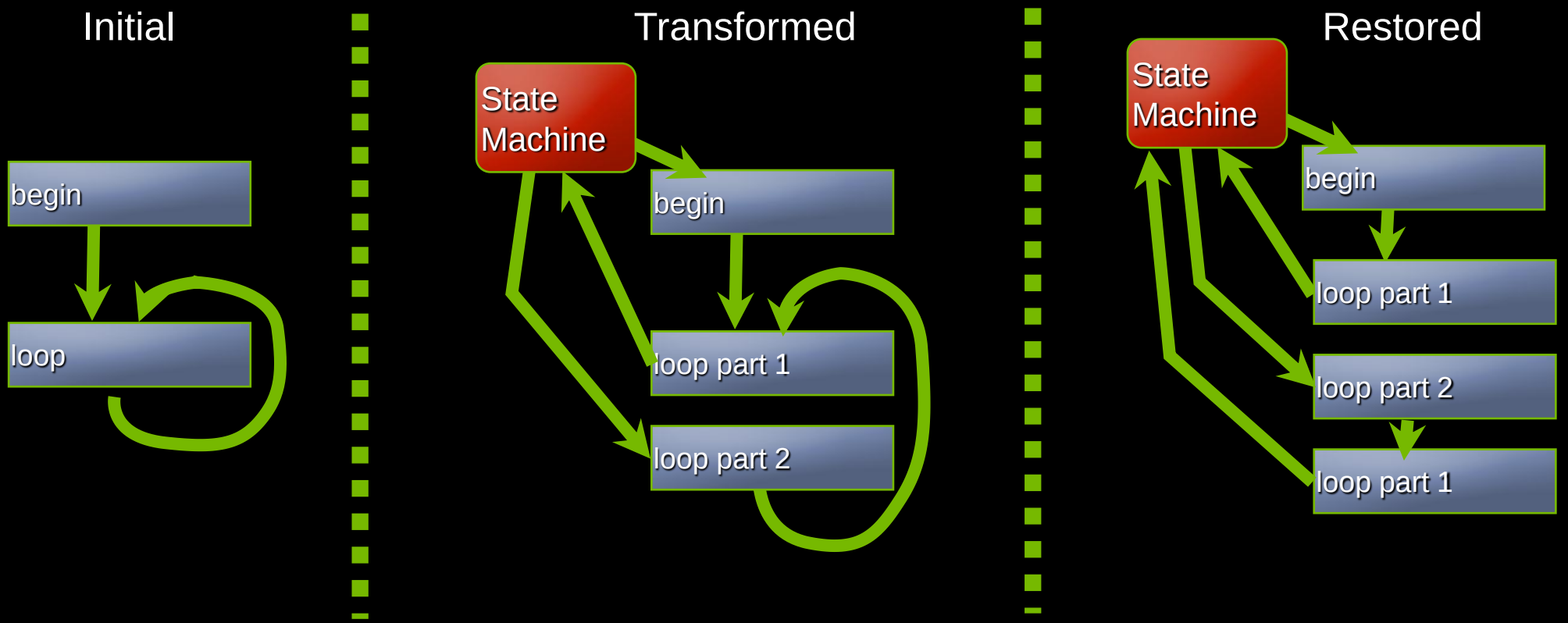
Step 5: Restore structured control flow

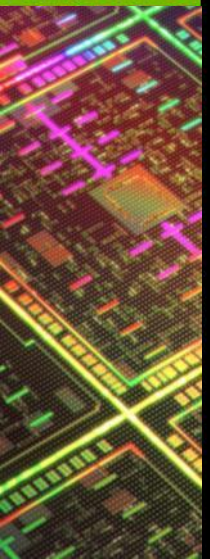
```
state 1:
  ld.const.u32    %node, [top_object+0];
  mov.s32          %i, 0;
loop:
  mov payload, %stack;
  save %i;
  bra mainloop;

state 2:
  restore %i;
  rematerialize %iend, %node;
  add.s32          %i, %i, 1;
  mov.u32          %iend, 5;
  setp.ne.s32      %predicate, %i, %iend;
  @%predicate bra loop;
  bra mainloop;
```

OptiX

```
state 1:
  ld.const.u32    %node, [top_object+0];
  mov.s32          %i, 0;
loop:
  mov payload, %stack;
  save %i;
  bra mainloop;

loop_copy:
  mov payload, %stack;
  save %i;
  bra mainloop;
state 2:
  restore %i;
  rematerialize %iend, %node;
  add.s32          %i, %i, 1;
  mov.u32          %iend, 5;
  setp.ne.s32      %predicate, %i, %iend;
  @%predicate bra loop_copy;
  bra mainloop;
```

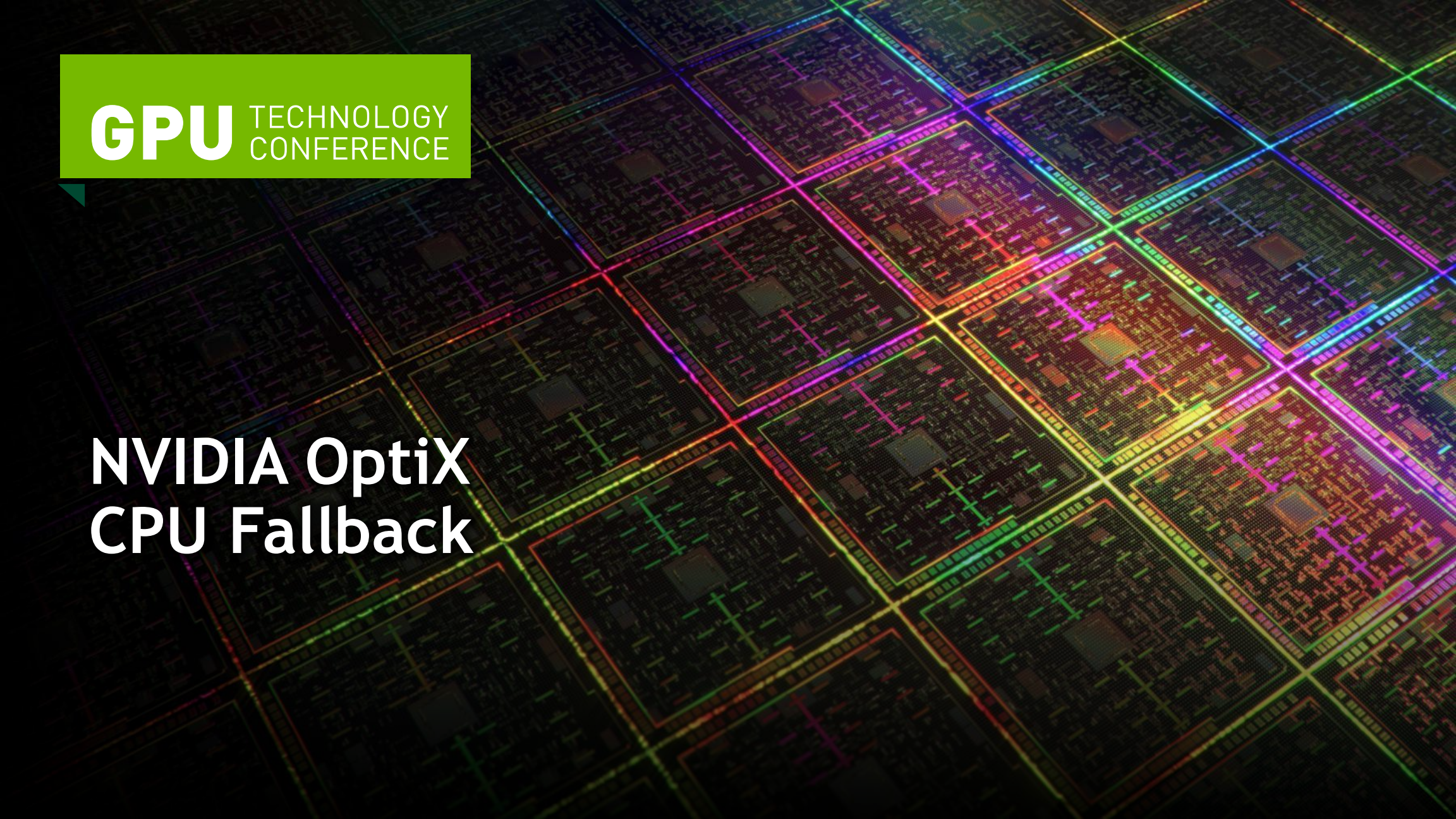
Graphical View





Other optimizations

- JIT Compiler gives opportunity for data-dependent optimizations
  - Elide unused transforms: up to 7%
  - Eliminate printing/exception code when not enabled: arbitrary
  - Reduce continuation size with rematerialization: arbitrary
  - Specialize traversal based on tree characteristics: 10-15%
  - Move small read-only data to textures or constant memory: up to 29%
- Traditional compiler optimizations
  - Loop hoisting, dead-code elimination, copy propagation, etc.
- Architecture-dependent optimizations
  - Different load-balancing, scheduling, traversal, V4 LD/ST

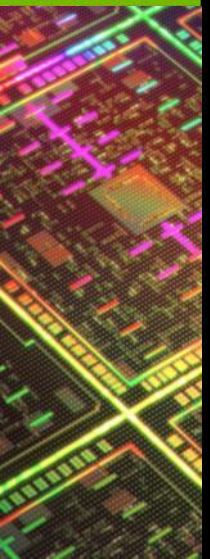

GPU TECHNOLOGY CONFERENCE
NVIDIA OptiX
CPU Fallback

CPU Fallback

- What is it?
- Why do it?
- How did we do it?

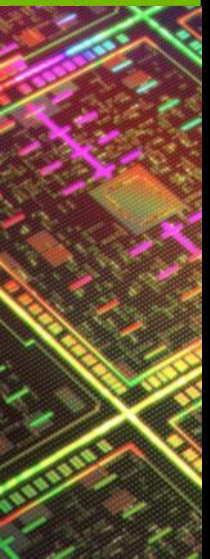
What is CPU Fallback?

- NVIDIA GPUs ROCK!



What is CPU Fallback?

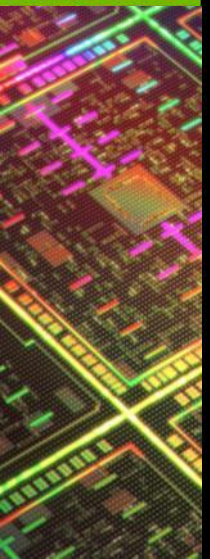
- NVIDIA GPUs ROCK!
- What if you aren’t fortunate enough to have one?



What is CPU Fallback?

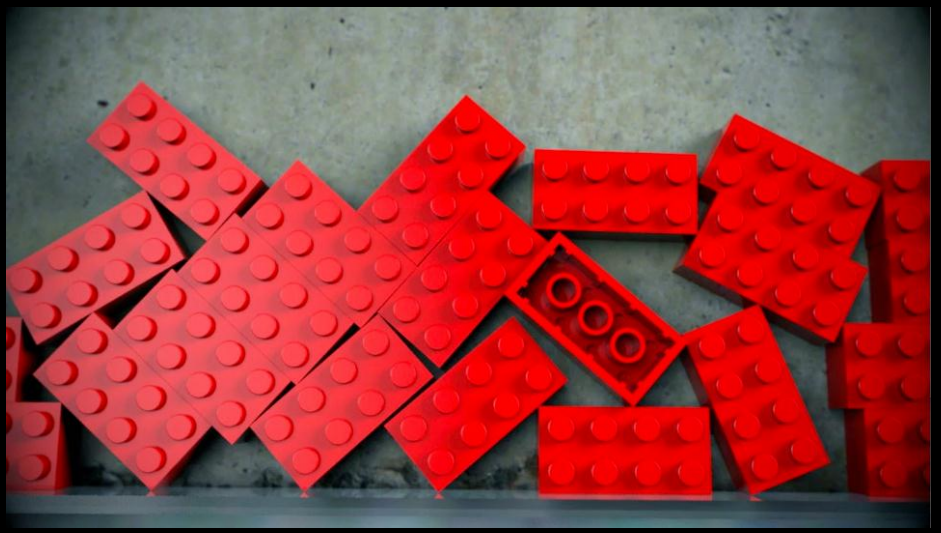
- NVIDIA GPUs ROCK!
- What if you aren't fortunate enough to have one?

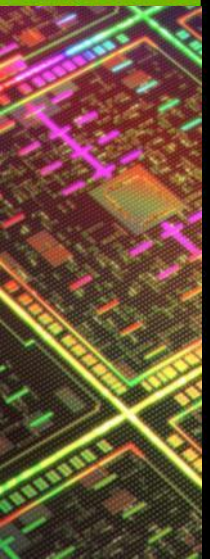
- Same executable with no changes
  - Run on NVIDIA hardware when present
  - Run on CPU when NVIDIA hardware or driver absent



Why do CPU Fallback?

- NVIDIA sells GPUs, not software
- OptiX is middle-ware
  - We depend on software vendors to integrate
  - Vendor software sales drive NVIDIA hardware sales
- Software vendors want OptiX
  - OptiX is an enabling technology
  - Still want their software to work for all customers

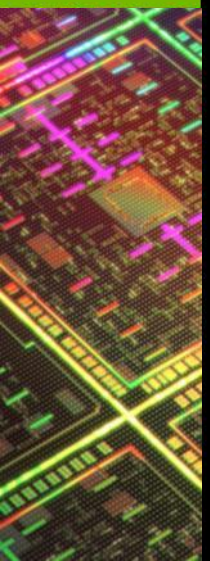


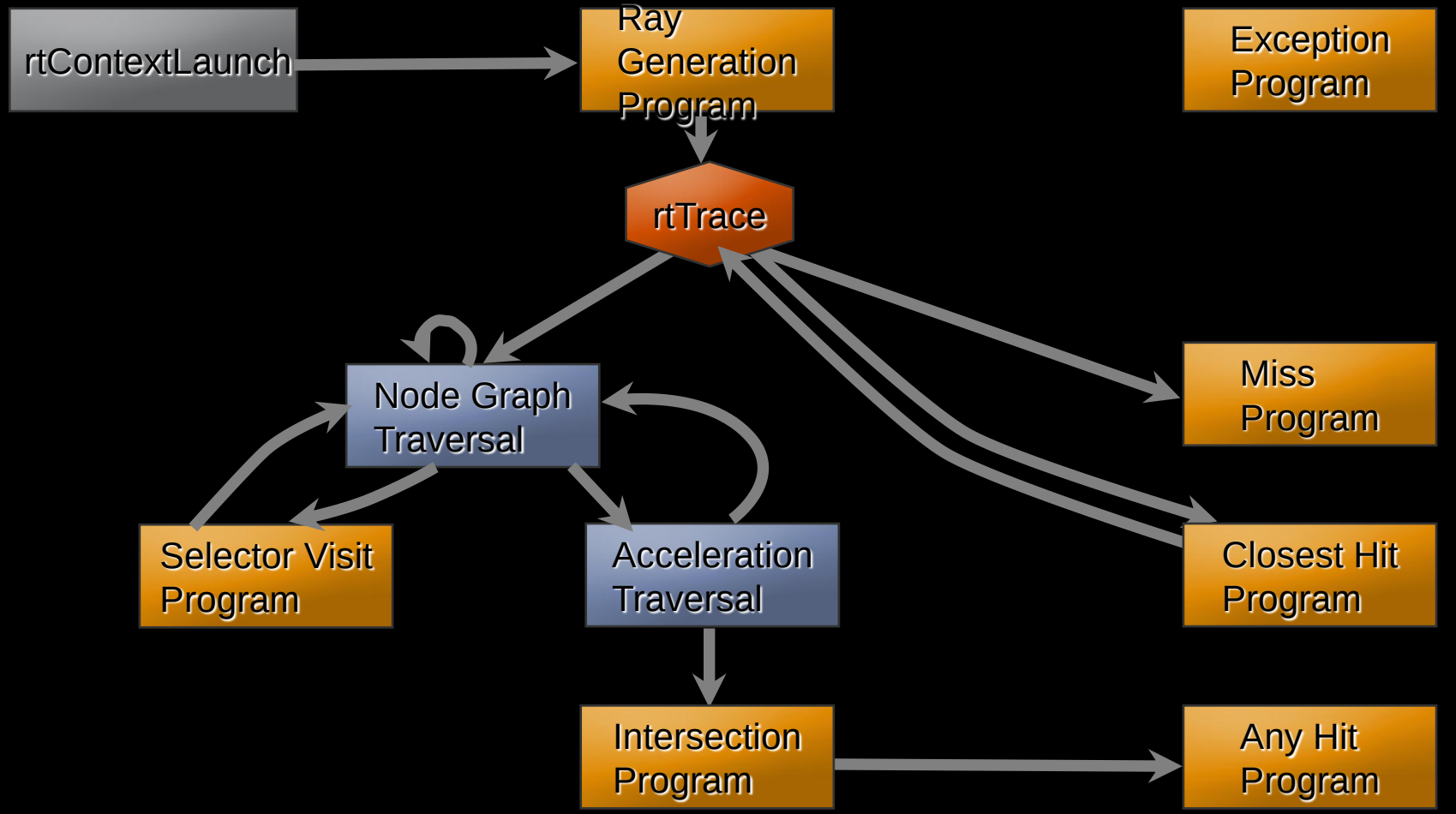


How did we do it?

- Short answer
  - Slight modifications to the OptiX generated code
  - Use GPU Ocelot to translate PTX to LLVM IR
  - Use LLVM to lower LLVM IR to x86
  - Run code on CPU
- Long answer …

Overview of OptiX Programming model



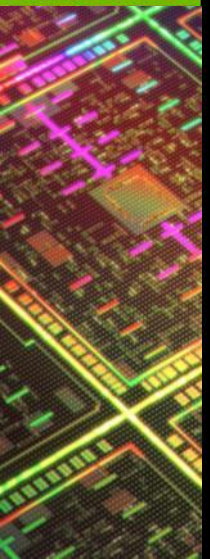


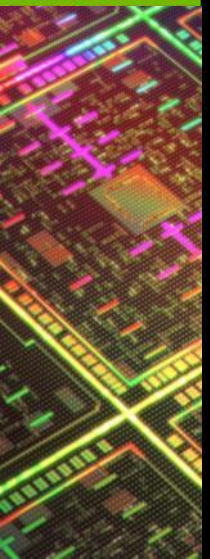
How do Optix Programs work?

```
shade()
{
  // A
  rtTrace(…);
  // B
}
```

```
switch (program) {
  case shade_A:
    bra shade_A_label;
  case shade_B:
    bra shade_B_label;
  case trace:
    bra trace_label:
```

```
shade_A_label:
  // A
  program = trace;
  return_program = shade_B;

shade_B_label:
  // B
  program = return_program;

trace_label:
  // trace
  program = return_program;
```



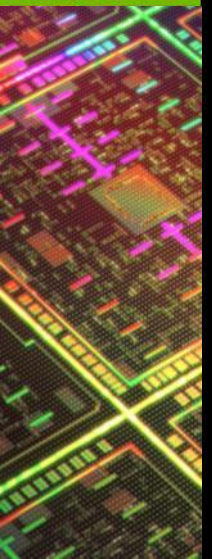


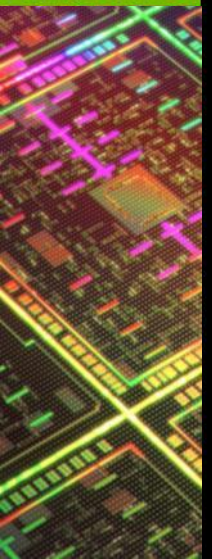
An example of OptiX GLUE for rtTrace

```
void rtTrace( Node node, Ray new_ray, void* prd) {
    pushState(); // Inserted by rewriteContinuations
    count rays
    current_ray = new_ray;
    current_prd = prd;

      if( raygen || exception program) {
        attribute_bottom = stack + stacksize;
      } else {
        attribute_bottom =
  min(current_attribute_frame,
  closest_attribute_frame)
      }

      closest_attribute_frame = attribute_bottom -
  attribute_frame_size;
      if(!miss or closest hit program)
        current_attribute_frame = attribute_bottom -
  attribute_frame_size * 2;

      if(stack_cur > smaller of current or
  closest_attribute_frame)
        goto stack overflow;

      terminate_closure =
  vpc_for_return_from_intersct | ((stack_cur -
stack_top) << VPC_SHIFT)

      current_transform_depth = 0;
      closest_transform_depth = 0;

      closest_instance = -1
      current_node = node;
      call current_node->visit_node();

      if(closest_instance!=-1) {
        saved_tmax = ray_tmax;
        current_instance = closest_instance;
        current_material = closest_material;
        call current_material->closesthit[ray_type];
      } else {
        call miss[ray_type];
      }
      popState(); // Inserted by rewriteContinuations
    }
```

Megakernel - vs - Individual Functions

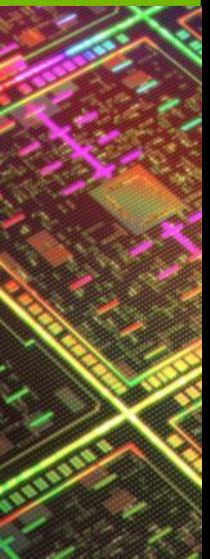
- Was about 8x faster than calling individual host functions
  - Small functions were hard for LLVM to optimize
  - Lots of calling back and forth between “host” and “user” code
- Glue code only lives in a single place
  - Fewer bugs
  - Better code maintenance
- This of course could change in the future…





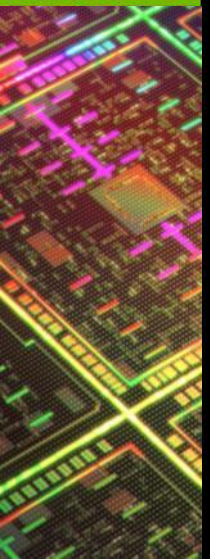
PTX to X86 Translation

- PTX features
  - Infinite register file
  - memory spaces (ld/st)
    - constant, local, shared, global
  - branch
  - predicated instructions
  - variety of special functions
    - cvt, bfe, bfi
    - sqrt, rsqrt, sin, cos, lg2, ex2, max, min



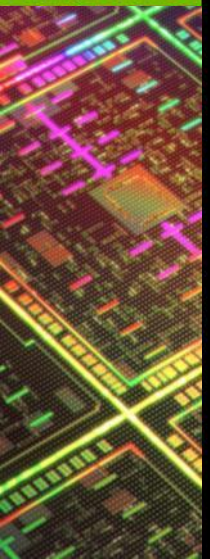
PTX to X86 Translation

- PTX features
  - Infinite register file
  - memory spaces (ld/st)
    - constant, local, shared, global
  - branch
  - predicated instructions
  - variety of special functions
    - cvt, bfe, bfi
    - sqrt, rsqrt, sin, cos, lg2, ex2, max, min



PTX to X86 Translation

- PTX features
  - Infinite register file
  - memory spaces (ld/st)
    - constant, local, shared, global
  - branch
  - predicated instructions
  - variety of special functions
    - cvt, bfe, bfi
    - sqrt, rsqrt, sin, cos, lg2, ex2, max, min



PTX CVT

Table 91. Data Movement and Conversion Instructions: cvt

cvt	Convert a value from one type to another.

Syntax	`cvt{.irnd}{.ftz}{.sat}.dtype.atype  d, a;  // integer rounding`
	`cvt{.frnd}{.ftz}{.sat}.dtype.atype  d, a;  // fp rounding`
	
	`.irnd  = { .rni, .rzi, .rmi, .rpi };`
	`.frnd  = { .rn,  .rz,  .rm,  .rp  };`
	`.dtype = .atype = { .u8, .u16, .u32, .u64,`
	`.s8, .s16, .s32, .s64,`
	`.f16, .f32, .f64 };`

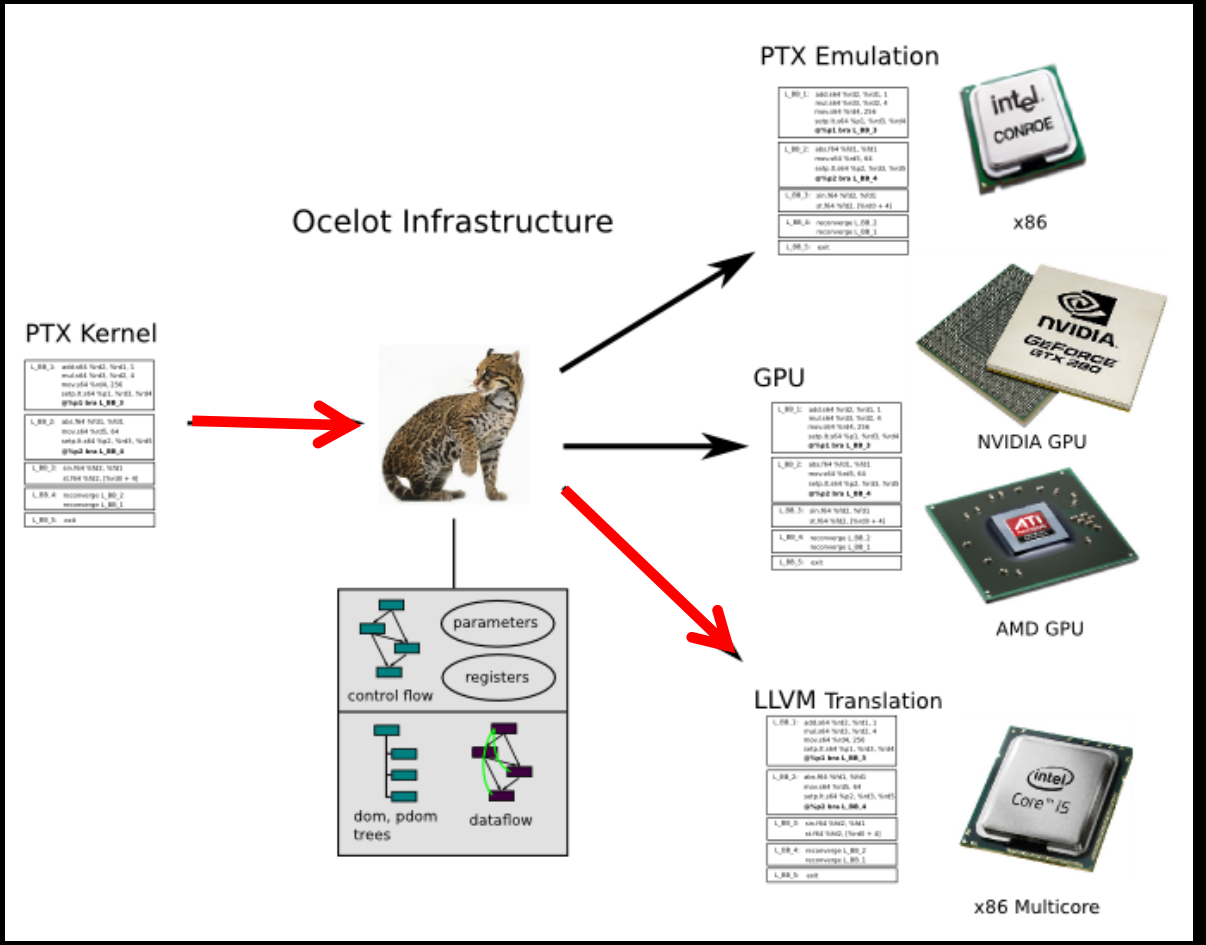
Description	Convert between different types and sizes.
Semantics	`d = convert(a);`
Integer Notes	Integer rounding is required for float-to-integer conversions, and for same-size float-to-float conversions where the value is rounded to an integer. Integer rounding is illegal in all other instances. Integer rounding modifiers: **.rni** round to nearest integer, choosing even integer if source is equidistant between two integers. **.rzi** round to nearest integer in the direction of zero **.rmi** round to nearest integer in direction of negative infinity **.rpi** round to nearest integer in direction of positive infinity Subnormal numbers: **sm_20:** By default, subnormal numbers are supported. For cvt.ftz.*dtype*.f32 float-to-integer conversions and cvt.ftz.f32.f32 float-to-float conversions with integer rounding, subnormal inputs are flushed to sign-preserving zero. **sm_1x:** For cvt.ftz.*dtype*.f32 float-to-integer conversions and cvt.ftz.f32.f32 float-to-float conversions with integer rounding, subnormal inputs are flushed to sign-preserving zero. The optional .ftz modifier may be specified in these cases for clarity. **Note**: In PTX ISA versions 1.4 and earlier, the cvt instruction did not flush single-precision subnormal inputs or results to zero if the destination type size was 64-bits. The compiler will preserve this behavior for legacy PTX code. Saturation modifier: **.sat** For integer destination types, **.sat** limits the result to MININT..MAXINT for the size of the operation. Note that saturation applies to both signed and unsigned integer types. The saturation modifier is allowed only in cases where the destination type's value range is not a superset of the source type's value range; i.e., the .sat modifier is illegal in cases where saturation is not possible based on the source and destination types. For float-to-integer conversions, the result is clamped to the destination range by default; i.e, **.sat** is redundant.
Floating Point Notes	Floating-point rounding is required for float-to-float conversions that result in loss of precision, and for integer-to-float conversions. Floating-point rounding is illegal in all other instances. Floating-point rounding modifiers: **.rn** mantissa LSB rounds to nearest even **.rz** mantissa LSB rounds towards zero **.rm** mantissa LSB rounds towards negative infinity **.rp** mantissa LSB rounds towards positive infinity A floating-point value may be rounded to an integral value using the integer rounding modifiers (see Integer Notes). The operands must be of the same size. The result is an integral value, stored in floating-point format. Subnormal numbers: **sm_20:** By default, subnormal numbers are supported. Modifier .ftz may be specified to flush single-precision subnormal inputs and results to sign-preserving zero. **sm_1x:** Single-precision subnormal inputs and results are flushed to sign-preserving zero. The optional .ftz modifier may be specified in these cases for clarity. **Note**: In PTX ISA versions 1.4 and earlier, the cvt instruction did not flush single-precision subnormal inputs or results to zero if either source or destination type was .f64. The compiler will preserve this behavior for legacy PTX code. Specifically, if the PTX ISA version is 1.4 or earlier, single-precision subnormal inputs and results are flushed to sign-preserving zero only for cvt.f32.f16, cvt.f16.f32, and cvt.f32.f32 instructions. Saturation modifier: **.sat** For floating-point destination types, **.sat** limits the result to the range [0.0, 1.0]. NaN results are flushed to positive zero. Applies to **.f16**, **.f32**, and **.f64** types.
Notes	Registers wider than the specified source or destination types may be used.
PTX ISA Notes	Introduced in PTX ISA version 1.0.
Target ISA Notes	**cvt** to or from **.f64** requires **sm_13** or later.

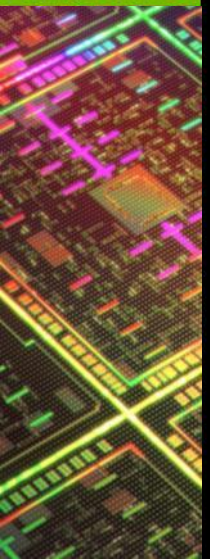
Examples	`cvt.f32.s32 f,i;`
	`cvt.s32.f64 j,r;  // float-to-int saturates by default`
	`cvt.rni.f32.f32 x,y; // round to nearest int, result is fp`
	`cvt.f32.f32 x,y;  // note .ftz behavior for sm_1x targets`

PTX to X86 Translation

- Currently uses open source GPU Ocelot library

http://code.google.com/p/gpuocelot/



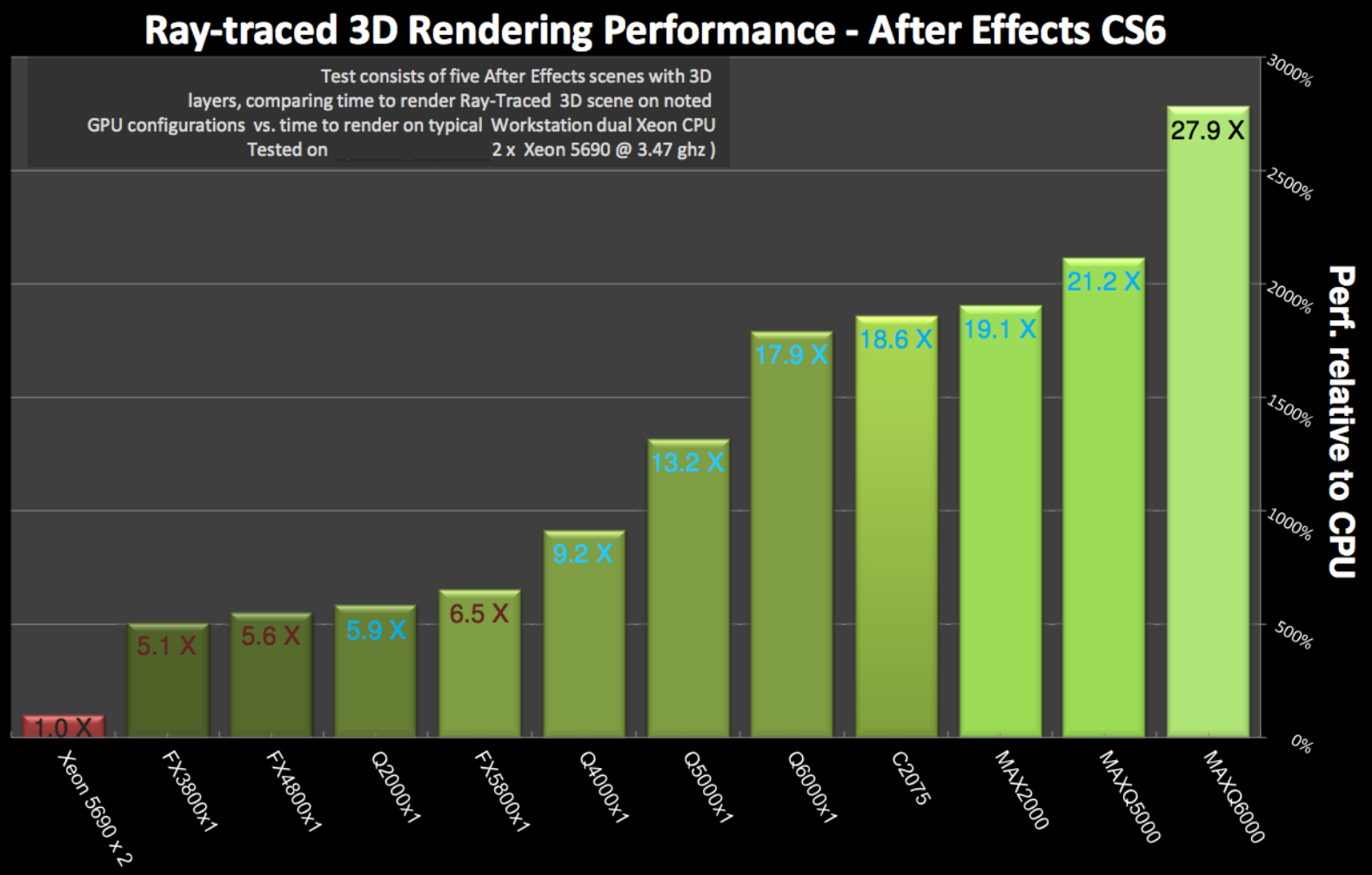


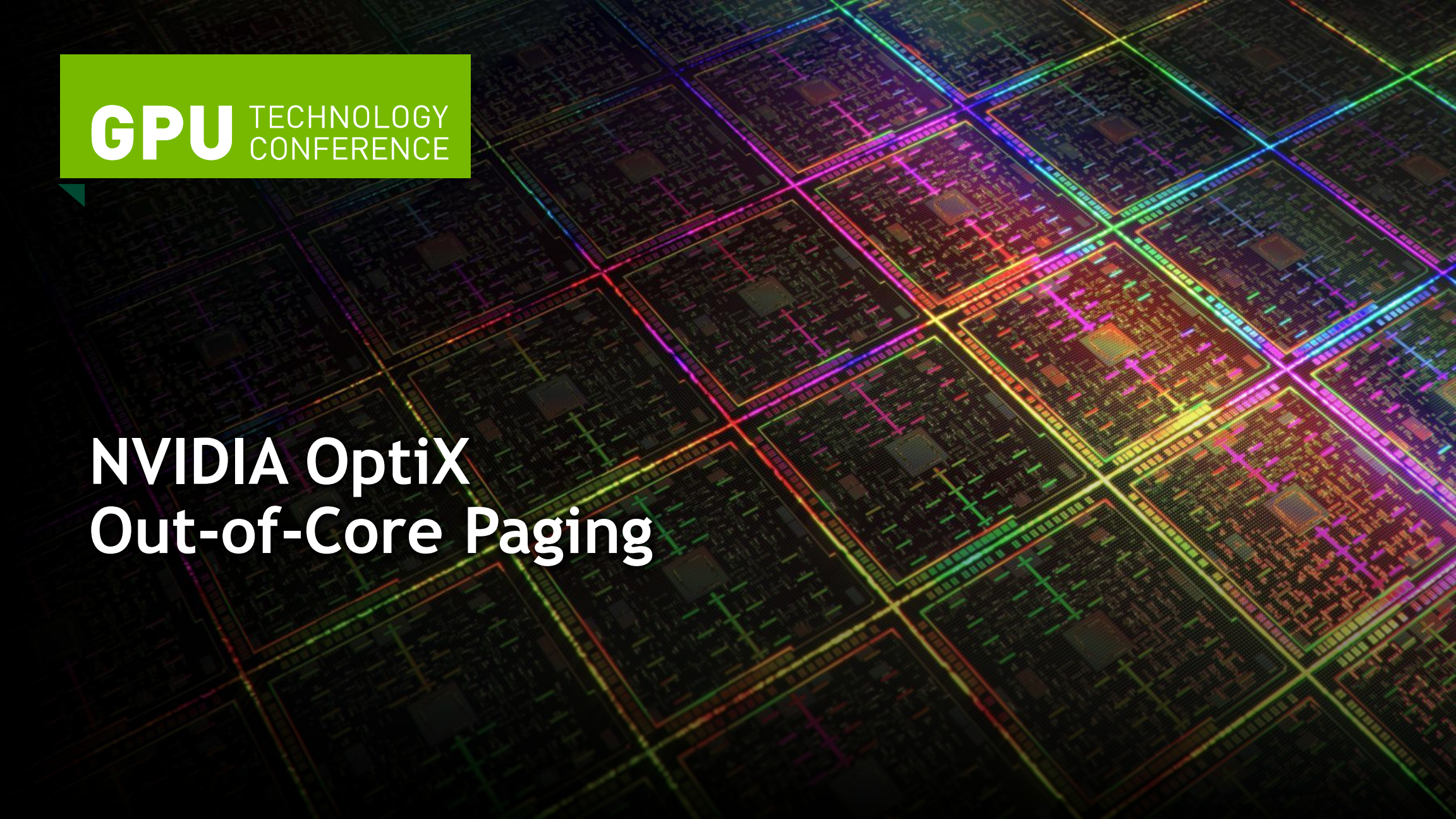
Translation example

```
max.ftz.f32 %f43, %f9, %f42;

%rt6655 = fcmp uno float 0x0, %r9687;
%rt6656 = select i1 %rt6655, float %r9644, float %r9687;
%rt6654 = fcmp ogt float %r9644, %r9687;
%rt6657 = select i1 %rt6654, float %r9644, float %rt6656;
%rt6658 = fcmp olt float %rt6657, 0x0;
%rt6659 = fsub float 0x0, %rt6657;
%rt6660 = select i1 %rt6658, float %rt6659, float %rt6657;
%rt6661 = fcmp olt float %rt6660, 0x3810000000000000;
%r9688 = select i1 %rt6661, float 0x0, float %rt6657;
```

- Future optimization could identify unneeded modifiers to reduce translated code cost

How well does it perform?

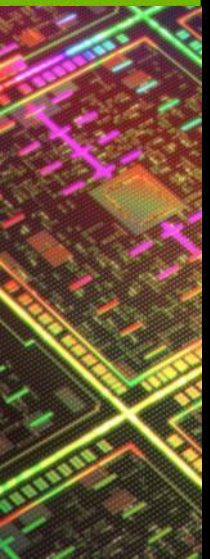




NVIDIA OptiX Out-of-Core Paging

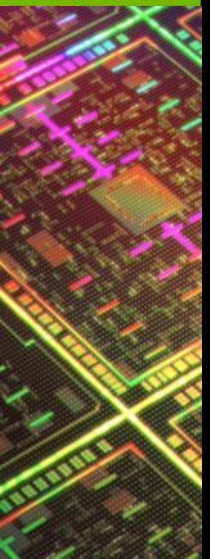
Paging

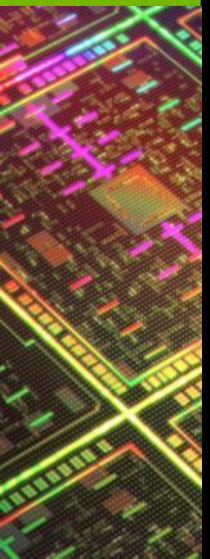
- Use cases:
  - Mildly oversubscribed:
    - (513MB dataset, 512MB card)
  - Largely oversubscribed:
    - (20GB dataset, 6GB card)
- Approach: Use OptiX Compiler to implement virtual memory system in OptiX kernel



Preparing for Paging

- Choose which buffers to page
  - Input buffers, too big for GPU, certain use cases like BVH data
- Allocate the page table
- Allocate a large page cache
- Compile megakernel for paging



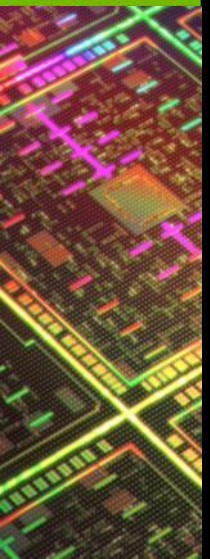


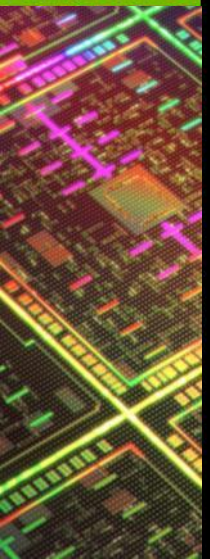
LD Rewrite

- Page translation
- On fault:
  - Mark page as requested
  - Save registers
  - Suspend thread
  - Get other work to do
- Clustering paged LDs
- Continuations

Kernel Launch Loop

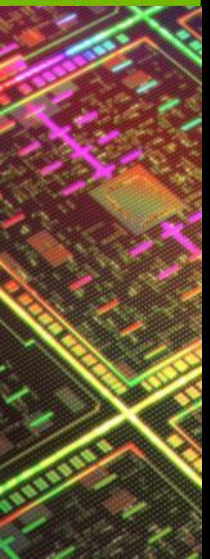
- While not all rays finished
  - Launch kernel
  - Choose page replacements
  - Upload requested pages





Filling Page Requests (on GPU)

- Scan for requested pages
- Choose victim pages based on LRU
  - Sort page table by time stamp
- Make list of page copies to perform
- Update time stamps of all accessed pages

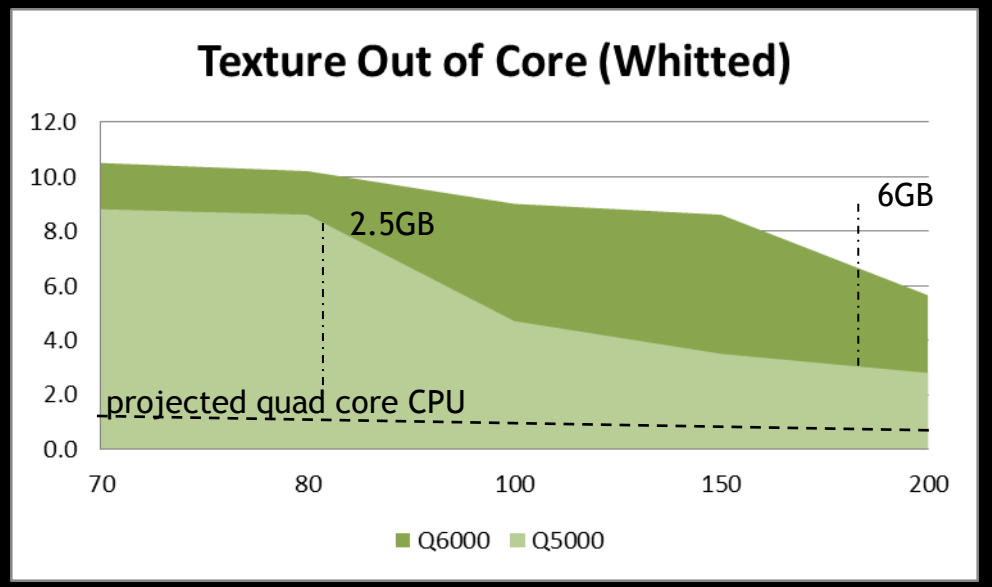


Page Copies

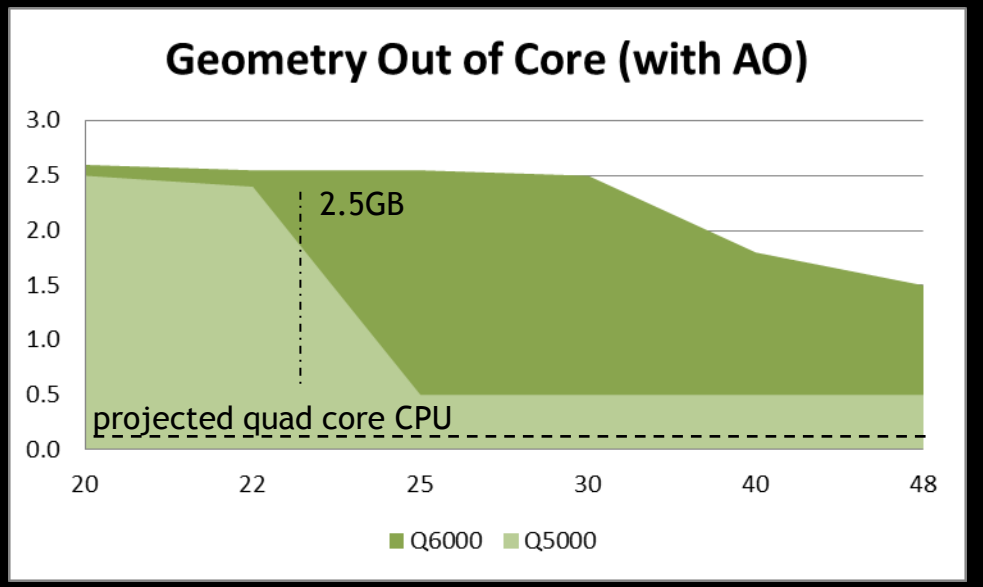
- Stage pages to be copied
  - Must be in pinned host memory
  - Use cudaHostRegister
  - Copy data into cudaAllocHost staging buffer
- Copy requested pages to GPU
  - cuMemCopyHtoD
  - Copy kernel

OptiX 2.5 Out of Core Performance

- Averaged results, as paging amount is view dependent



# of 4k Images



Millions of Textured & Smoothed Faces

Quadro 6000 = 6GB on board memory

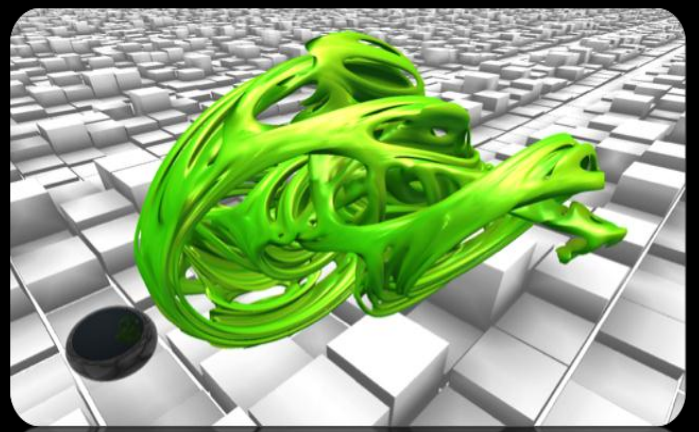
Quadro 5000 = 2.5GB on board memory

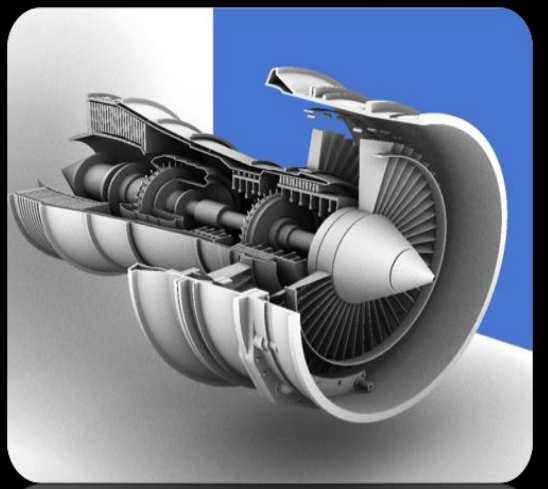
GPU TECHNOLOGY CONFERENCE

Thanks for Attending!

OptiX SDK

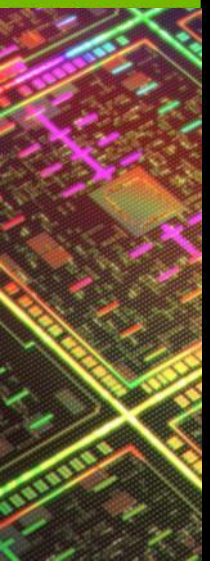
- Free to acquire and use: Windows, Linux, Mac
- http://developer.nvidia.com

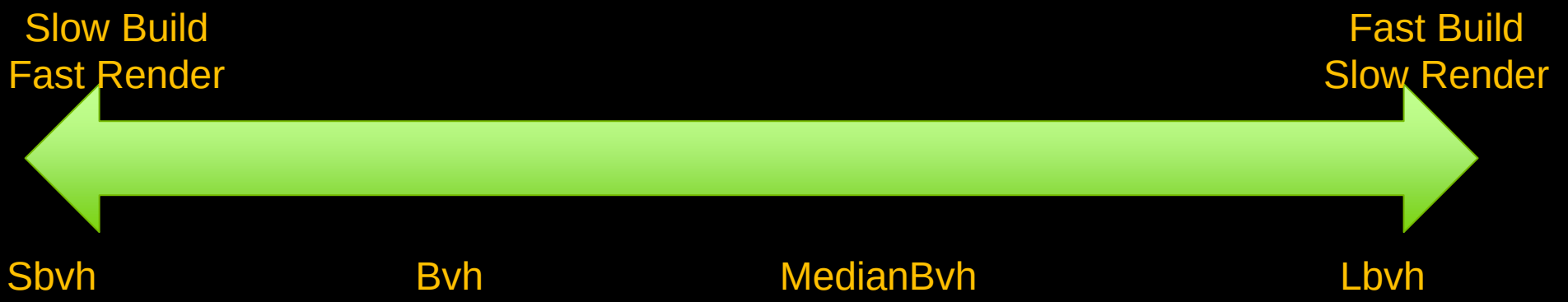




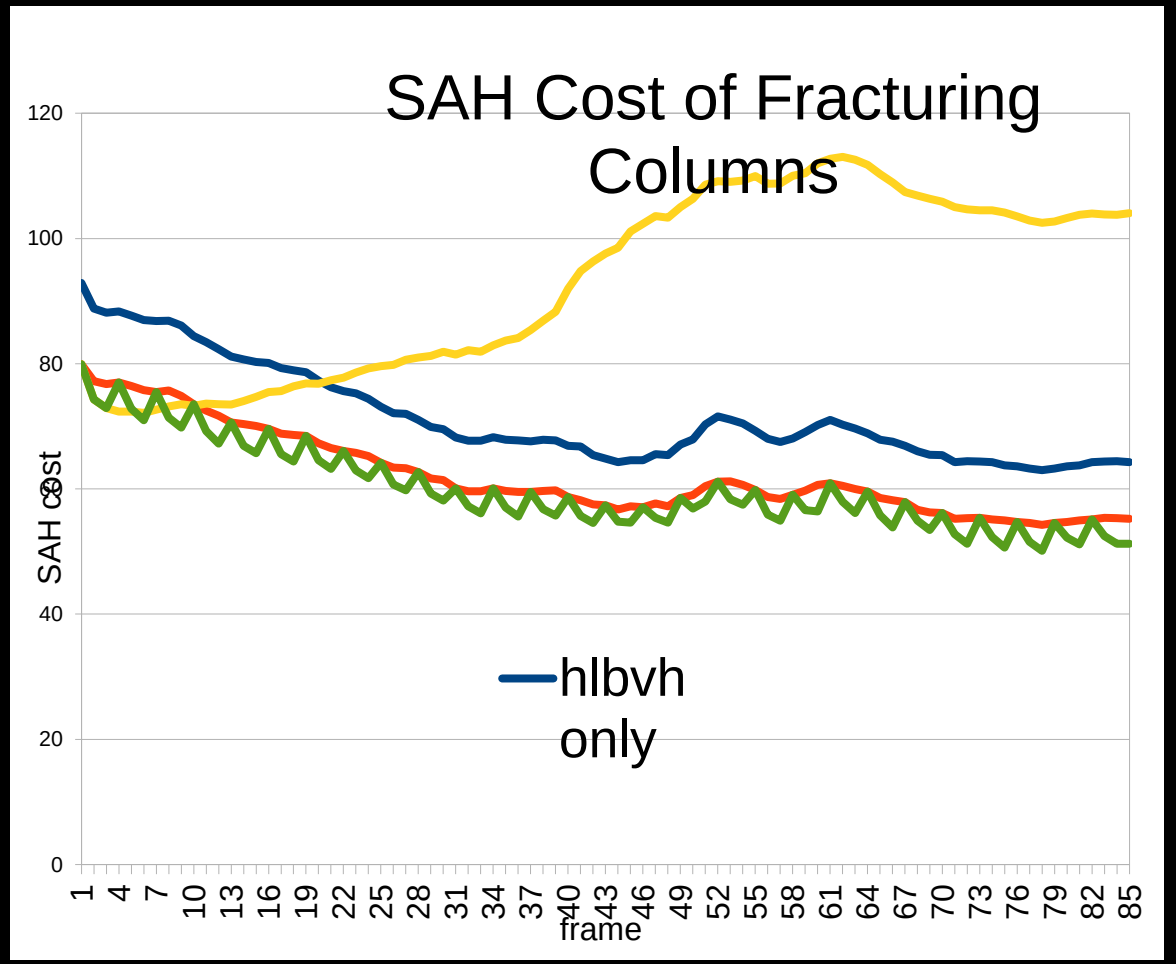
Acceleration Structures++

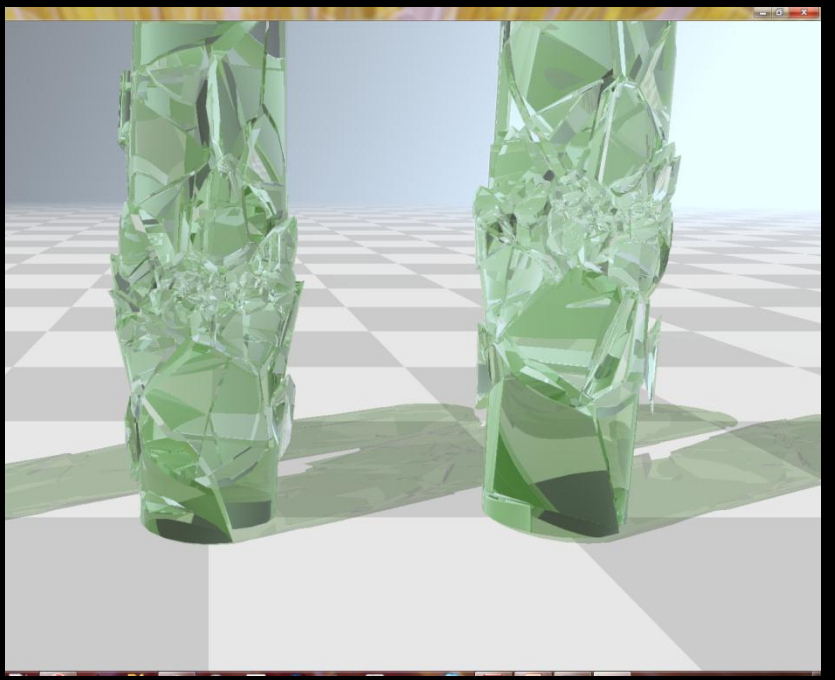
- "Sbvh" is up to 8X faster
- "Lbvh" is extremely fast and works on very large datasets
- BVH Refinement optimizes the quality of a BVH
  - Smoother scene editing
  - Smoother animation





BVH Refinement





CUDA-OptiX Interoperability

- Share a CUDA context between OptiX and CUDA runtime
- Share buffers on one device without memory copies
- Copy buffers from device to device peer-to-peer
  - Avoid round-trip through host