
NVIDIA
Advanced Rendering
and GPU Ray Tracing
SIGGRAPH 2012

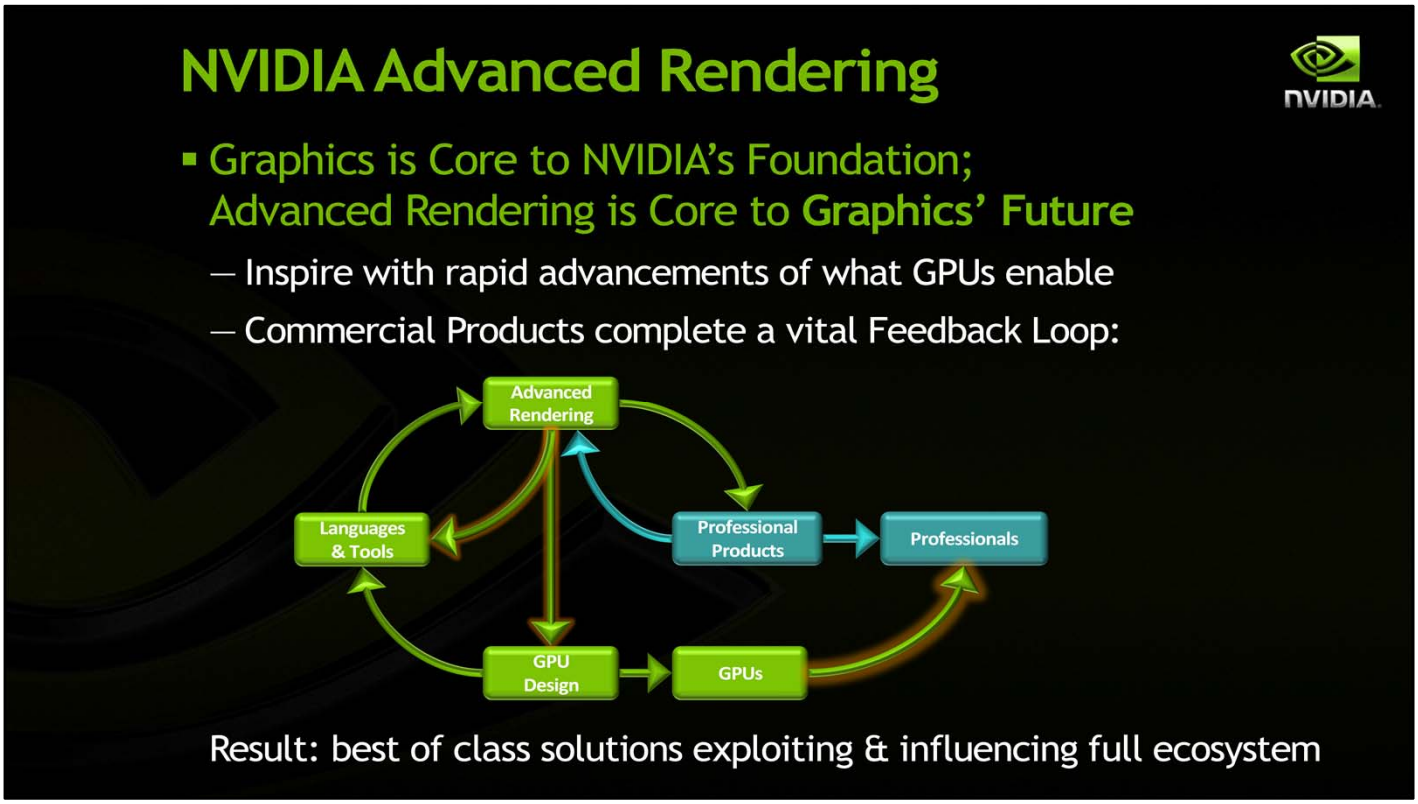
Agenda

1. Brief Introduction
   (with Phil Miller, Advanced Rendering product management)
2. Progress in Advanced Rendering (iray)
3. GPU Ray Tracing Basics
4. Introduction to OptiX
5. What's coming next in OptiX
   (with David McAllister, OptiX development manager)

NVIDIA Ray Tracing Options

- **CUDA** – language and computing platform
  - The basic choice for building *entirely custom solutions from scratch*
- **OptiX** – middleware for ray tracing developers
  - Good choice for developers *with domain expertise* building *custom solutions* which prefer *leaving GPU issues to NVIDIA*
- **iray & mental ray** – licensed rendering products
  - Good choice for companies wanting a *ready-to-integrate solution* which is *maintained and advanced for them*

NVIDIA Advanced Rendering



NVIDIA Rendering Options

- **mental ray**
  focusing on the needs of **Film Production**
  available StandAlone and licensed for integration

- **iray**
  focusing on the needs of **Design**
  licensed for integration

- **OptiX ray tracing framework**
  focusing on **general** GPU ray tracing development
  free to acquire and deploy

NVIDIA Iray - in use

images courtesy of Delta Tracing

full image gallery removed to spare download times

NVIDIA Iray Integration Framework

- For Software Developers wanting to add physically based, easy to use, iray rendering to their applications

- Procedure:
  1. Register your interest at www.mentalimages.com
     http://www.mentalimages.com/products/iray/iray-integration-framework/software-download.html
  2. NVIDIA reviews application, and grants access to SDK
  3. Integrate the SDK within your Application
  4. Once satisfied, obtain a commercial license from NVIDIA

NVIDIA Iray for End Users

- Iray enabled products:
  - Autodesk 3ds Max & 3ds Max Design
  - Dassault Systèmes Catia V6
  - Bunkspeed SHOT, MOVE, PRO
- **Recent features awaiting application integration:**
  - Multi-Layer BSDF Materials
  - Matte Objects
  - Motion Blur

NVIDIA Iray 3 - Released

- Includes:
  - New lighting algorithms for greater accuracy and superior caustics in more challenging lighting situations
  - Render buffer support (diffuse, specular, UVW's, ID's, etc.)
  - Flexible cluster management
- Initial Light Path Expression support now, final later this year
  - Extensive rendering pass control
  - Allows flexible post processing of iray production renderings

Extending Iray's Interactive Reach

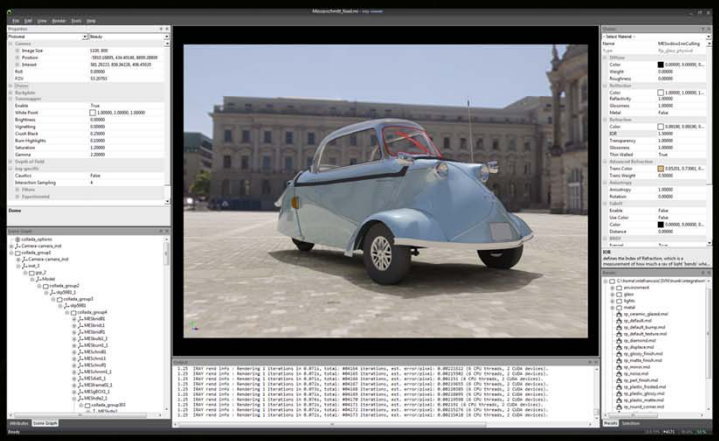
- Currently In Development, and available later this year
- Multiple rendering modes, providing a quality/speed continuum

iray "real-time"	iray "interactive"	iray "photoreal"
120 FPS ◄——► 15 FPS*	20 FPS ◄——► 0.5 FPS	10 FPS ◄——► Minutes
Stereo Game Title Quality Multi-Pass Effects Raster AO Soft Shadows, etc.	Accurate Reflections Accurate Shadows Soft Shadows Glossy Reflections Multi-Bounce Diffuse, etc.	Degraded Simplified Uncompromised Quality Increased Flexibility
Strength: Very High Resolutions	No / Minimal Noise while Interacting	Physically Based / Easy to Use
Weakness: Very Approximate	Not Physically Based	Noisy while Resolving

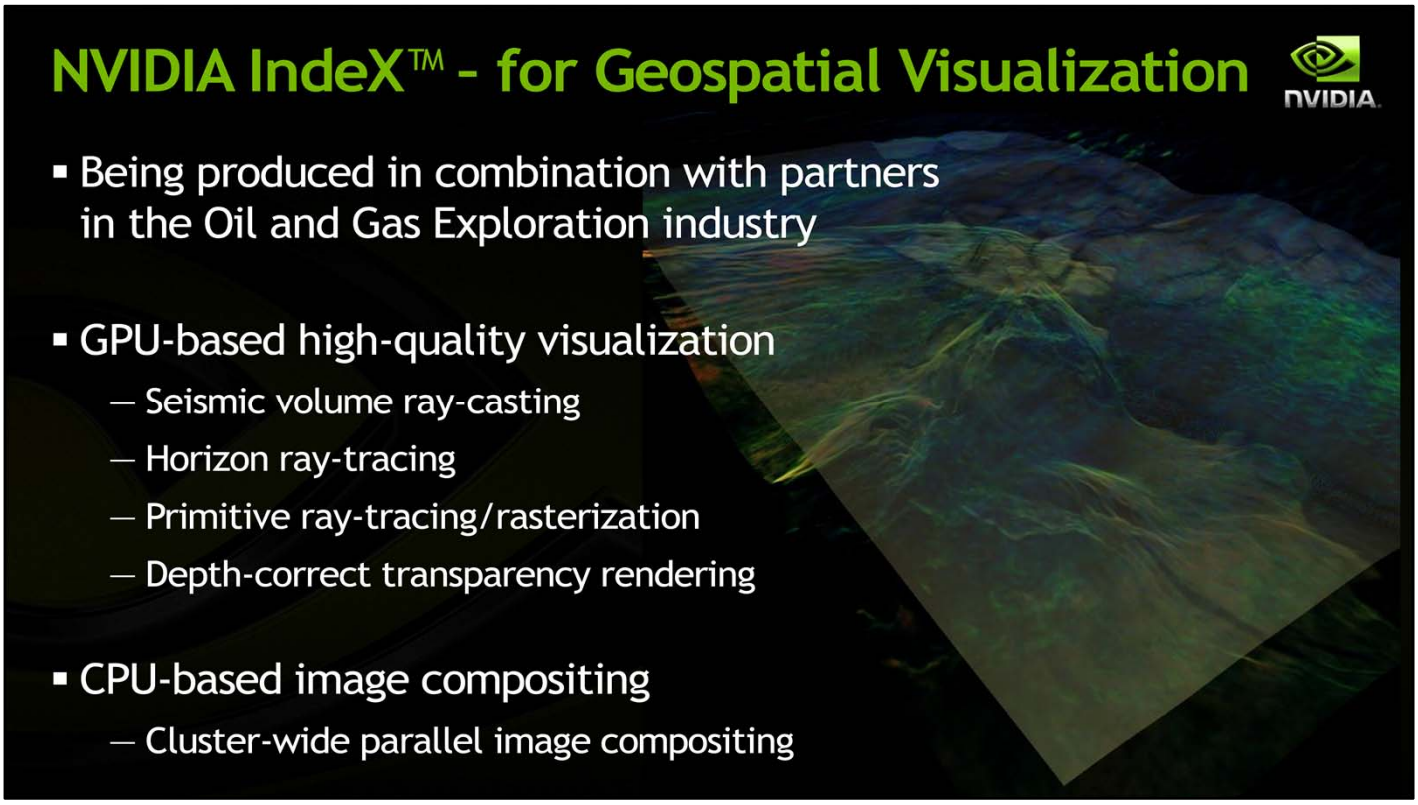
- APIs for which mode to use, with what features, what to do on mouse-up, etc. enable custom personalities for behavior and look

NVIDIA Iray – Progress Demo

- Showing IView – a test harness application from the SDK
- Today showing:
  - Iray Photoreal & Interactive
  - Sharing of materials
    - Common description
    - Physically based
    - Provides a stable target for matching without any algorithm knowledge
    - Easy for end users to manipulate

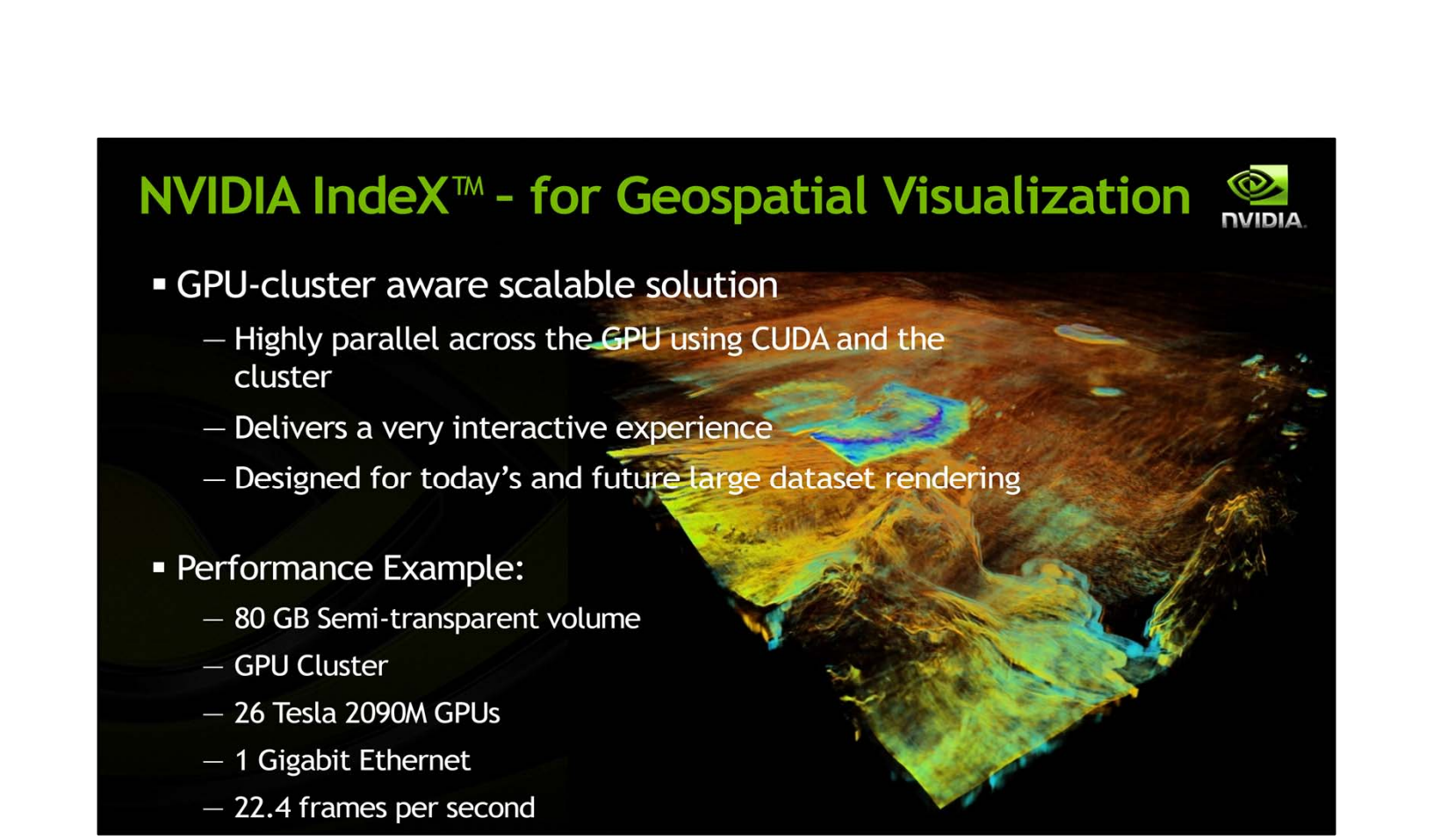


NVIDIA IndeX™ – for Geospatial Visualization



- Being produced in combination with partners in the Oil and Gas Exploration industry
- GPU-based high-quality visualization
  - Seismic volume ray-casting
  - Horizon ray-tracing
  - Primitive ray-tracing/rasterization
  - Depth-correct transparency rendering
- CPU-based image compositing
  - Cluster-wide parallel image compositing

NVIDIA IndeX™ - for Geospatial Visualization



- GPU-cluster aware scalable solution
  - Highly parallel across the GPU using CUDA and the cluster
  - Delivers a very interactive experience
  - Designed for today's and future large dataset rendering
- Performance Example:
  - 80 GB Semi-transparent volume
  - GPU Cluster
  - 26 Tesla 2090M GPUs
  - 1 Gigabit Ethernet
  - 22.4 frames per second

General GPU Ray Tracing

- Topics relating to most GPU ray tracing applications

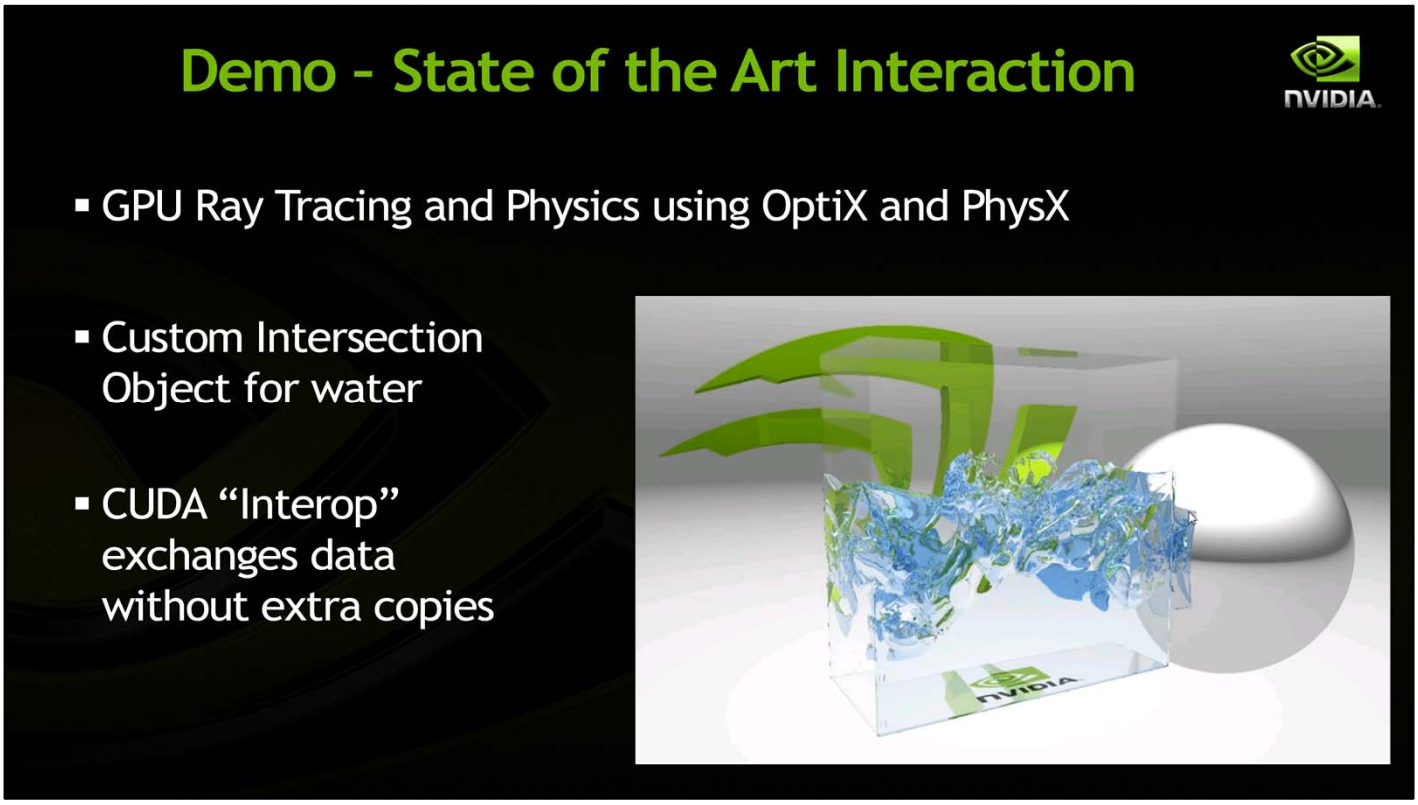
GPU Ray Tracing Myths

1. The only technique possible on the GPU is “path tracing”
   FALSE: Many techniques have been implemented
2. Only Professional GPUs can do ray tracing
   FALSE: GPU computing languages run on all NV GPUs
3. A GPU farm is more expensive than a CPU farm
   FALSE: Much better Perf/$; with better Perf/Watt on Kepler
4. A GPU isn’t that much faster than a good CPU
   FALSE: A single GPU is typically 4-12X a quad-core CPU
5. GPU Ray Tracing is very difficult
   Very Possibly: OptiX speeds both ray tracing and GPU development
6. Scenes must fit into GPU memory – and that’s finite
   Not Always: Panta Ray, CentiLeo, OptiX 2.5 paging, NVIDIA IndeX

GPU Ray Tracing Facts

1. GPUs can accommodate most any ray tracing technique
2. Compute, and thus ray tracing, works on all NVIDIA GPUs
3. GPUs have superior performance (and maintenance) costs vs. CPUs
4. A single GPUs is considerably faster than multiple CPUs
5. OptiX makes both Ray Tracing and GPU development easier
6. Scenes can exceed GPU memory with OptiX 2.5 (up to system RAM) and with custom approaches

Demo - State of the Art Interaction



Commercial GPU Ray Tracing

▪ iray	CUDA C, C Runtime	
▪ V-Ray RT	CUDA C, Driver API and OpenCL	
▪ Arion	CUDA C, Driver API	
▪ Octane	CUDA C, Driver API	
▪ finalRender	CUDA C, C Runtime	
▪ LuxRender (open source)	OpenCL	
▪ CentiLeo	CUDA C, driver API	Out of Core
▪ Panta Ray (Weta)	CUDA C, driver API	Massive Out of Core
▪ OptiX (2.5)	CUDA C, driver API & PTX	(Out of Core)
▪ Adobe After Effects CS6	OptiX API	" "
▪ Custom OptiX, Works Zebra, etc.	OptiX API	" "
▪ mental ray 3.11 (nearing Beta)	OptiX API	" "

GPU Ray Tracing Similarities – Performance

- Single GPU Ray Tracing Speed
  - Usually linear to GPU cores and Core Clock – *for a given GPU generation*
  - Gains between GPU generations often vary per application / technique
- Multi-GPU Ray Tracing Speed
  - Solution dependent, Common in Renderers, OptiX supports by default
  - Scaling efficiency varies by solution;
    slow techniques usually scale better than fast ones (e.g., AO vs. Whitted)
- Cluster Speed (multi-machine rendering)
  - Solution dependent, capabilities vary. OptiX doesn't, iray does

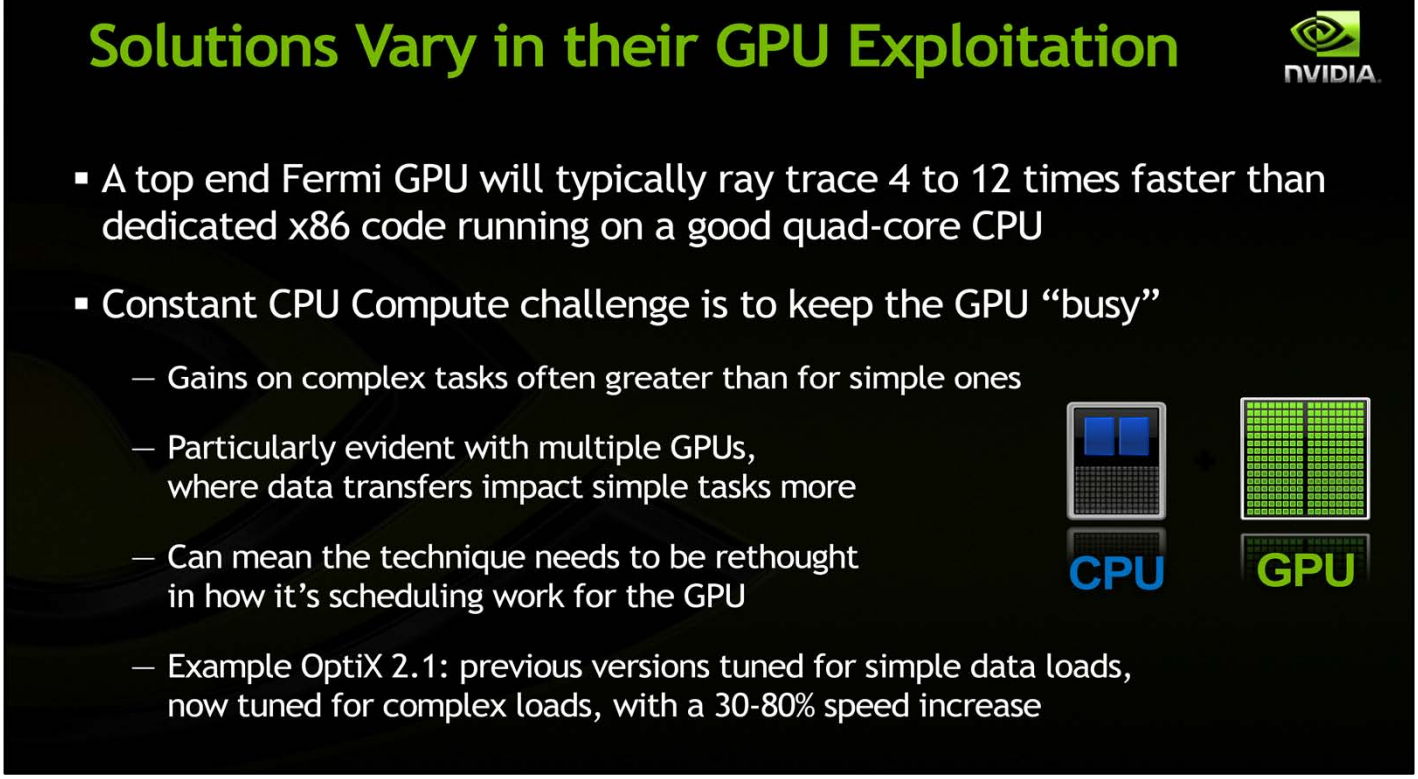
GPU Ray Tracing Similarities – Hardware

- "SLI" configuration is not needed for multi-GPU usage
- Nearly all renderers are Single Precision
- ECC driver choice (error correction) – **NOT** Recommended
  - No Accuracy Benefit; Slows Performance, Reserves ½ GB on a 3 GB board
- Windows 7 is a bit slower than Windows XP or Linux
- GPU memory size is often key
  - Entire scene must usually fit within GPU memory – to work AT ALL
  - Multiple GPUs can NOT "pool" memory; entire scene must fit onto each
  - If Out-of-Core is supported, it's much slower than fitting in memory
- Consumer GPUs aren't designed for constant "data center" usage

GPU Ray Tracing Similarities - Interaction

- GPU Computing (Ray Tracing) competes with system graphics
  - GPUs are still singularly focused: Compute **or** Graphics - not simultaneous
  - Often the **single biggest** design challenge for interactive app's
- Careful Application Design is needed to achieve balanced interaction
  - Gracefully stopping for user interaction and when app doesn't have focus
  - Controlling mouse pointers in the ray tracing app
- Or use Multi-GPU
  - One GPU for graphics, additional GPU(s) for compute (Ray Tracing)
  - Becoming mainstream with NVIDIA Maximus = Quadro + Tesla(s)

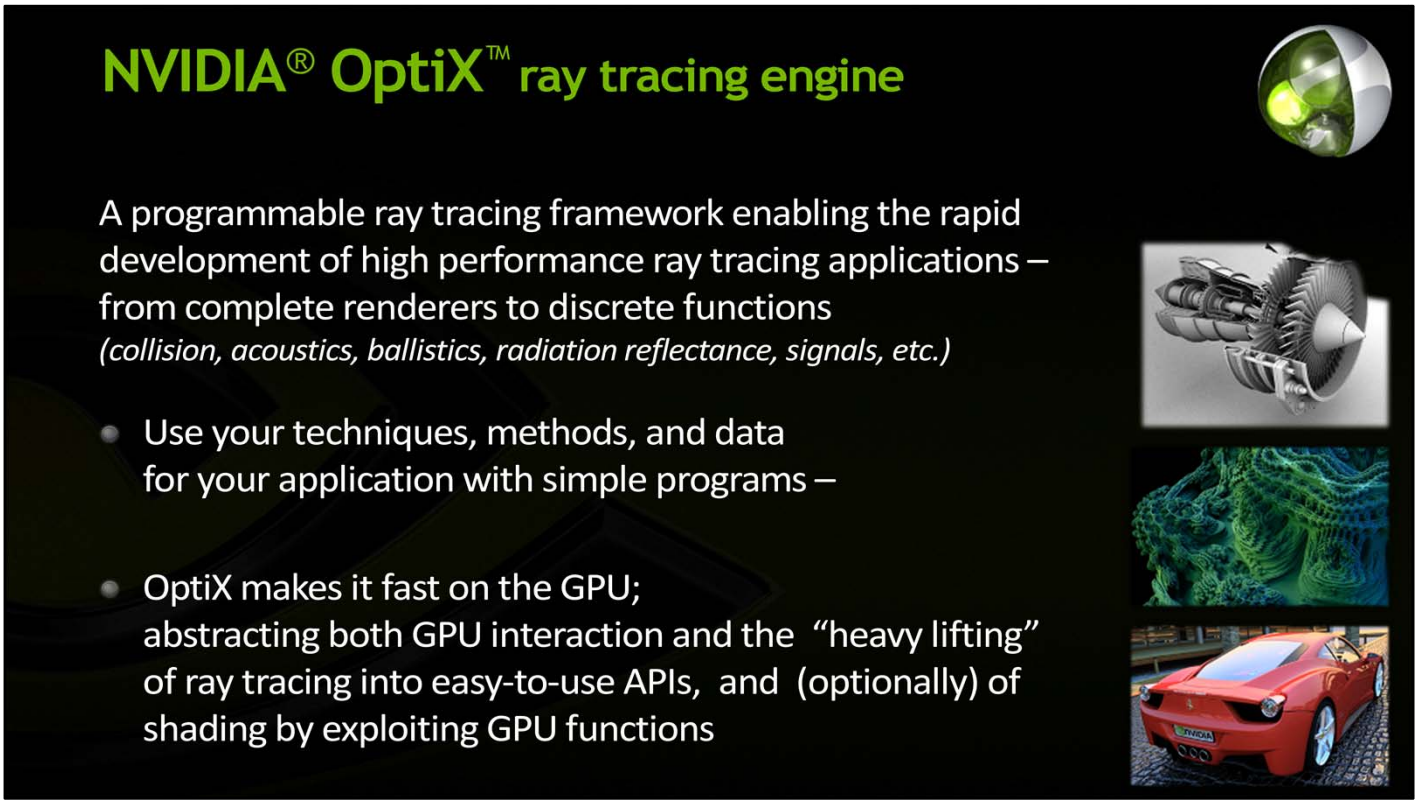
Solutions Vary in their GPU Exploitation



- A top end Fermi GPU will typically ray trace 4 to 12 times faster than dedicated x86 code running on a good quad-core CPU
- Constant CPU Compute challenge is to keep the GPU "busy"
  - Gains on complex tasks often greater than for simple ones
  - Particularly evident with multiple GPUs, where data transfers impact simple tasks more
  - Can mean the technique needs to be rethought in how it's scheduling work for the GPU
  - Example OptiX 2.1: previous versions tuned for simple data loads, now tuned for complex loads, with a 30-80% speed increase

Multi-GPU Considerations for Development

- Differing GPUs can mean different Compute capabilities
  - Not just between architectures (e.g., Fermi vs. Kepler) but sometimes within an architecture (e.g., GF100 vs. GF104)
  - Either insist on HW consistency from users, program to lowest denominator, or have multiple code paths
- TCC (Tesla Compute Cluster) mode for Windows
  - Default driver mode for new C-Class Tesla's (C2075 and newer)
  - Compute-only mode; GPU no longer a Windows graphics device
  - Should have parity with WDM driver with CUDA 5.0 (soon)



The OptiX engine works purely in compute.

OptiX is not a renderer, or even tied to rendering – it's a programmable ray tracing pipeline – much like OpenGL is for raster graphics.

It's currently being used in offline rendering, interactive rendering, and tasks that never produce an image – like collision detection, way-finding, acoustics, ballistics, sonar, where to place cell towers – when ever you're tracing rays.

OptiX is easy to develop with, and extremely flexible

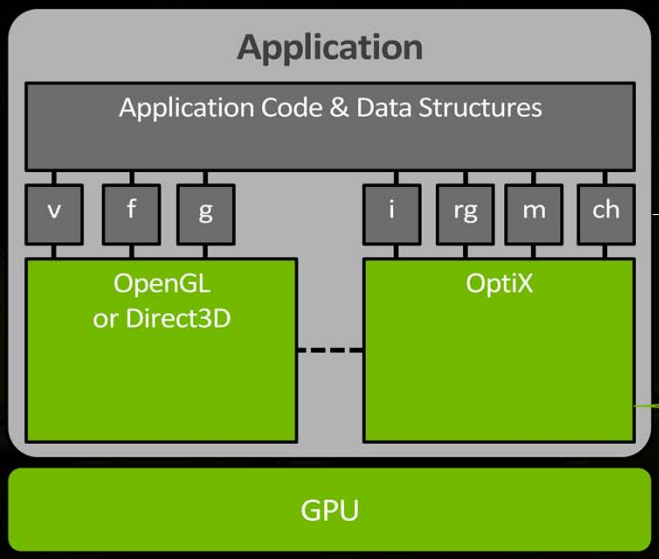
It combines easily with OpenGL or Direct3D for hybrid possibilities that can enhance any viewport

It's highly programmable, allowing you to process custom surfaces (algorithms, primitives, patches, NURBS) and have custom ray data – the "ray payload" – which is what allows any application type.

In taking care of making ray tracing fast, OptiX allows developers to concentrate on technique.

OptiX is good example of engines delivering the latest NVIDIA capabilities – giving a near 4X speed-up to applications as they run on GF100 (Fermi) GPUs.

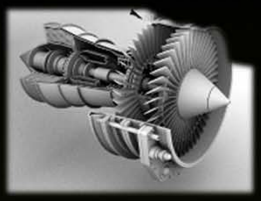
OptiX - similar to OGL in "Approach"



- C-based Shaders/Functions (minimal CUDA exp. reqd.)
  - Small, Custom Programs
  - Acceleration Structures Build & Traversal
- Optimal GPU parallelism and Performance
- Memory Management
- Paging

NVIDIA® OptiX™ ray tracing engine

- **Optimal performance**, from unique insights and methods for the latest GPU capabilities



- Easy to use, single ray programming model
- Supports custom ray generation, material shading, object intersection, scene traversal, ray payloads

- Programmable intersection for custom surface types (procedurals, patches, NURBS, displacement, hair, fur, etc.)
- No assumptions on technique, shading language, geometry type, or data structure

OptiX - in Use

+3k downloads per version

Adobe

LIGHTWORKS
Rendering Realism

WORKS ZEBRA

AUTHOR

ASPECTS

ARTISAN

TOYOTA

NEED FOR SPEED

Privately being used at companies doing:

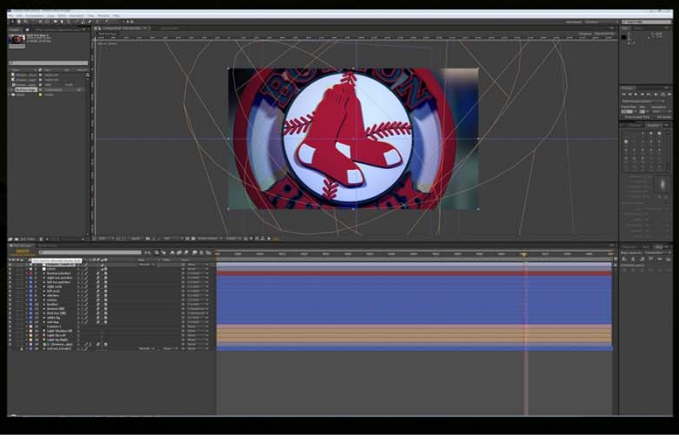
- Content creation tools
- Post production
- Next-Generation Gaming
- Massive On-Line Player Games and Services
- Acoustics
- Ballistics
- Multi-Spectral Simulation
- Radiation & Magnetic Reflection

Includes companies like: LEGO, Dolbe, CCP, Lockheed Martin, etc.

Adobe After Effects CS6 - using OptiX

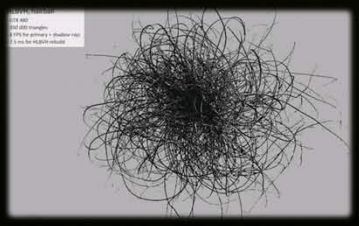
New 3D compositing with ray traced production renderer

- Built from scratch, in 1 release cycle
- 100% OptiX - no x86 code
- Includes CPU Fallback
  - Via LLVM in OptiX
  - Currently unique to Adobe



OptiX - Rapid Evolution

- Version 1, November 2009
  in use across many markets
- Version 2, August 2010
  exploited Fermi architecture for 2-5X speed increase
- Version 2.1, January 2011
  64-bit PTX, with +50% perf. on complex techniques, initial CPU fallback
- Version 2.5, April, 2012
  memory paging, GPU accel. structure build
- Version 2.6, very soon
  initial Kepler support & core improvements
- Version 3, Beta very soon – Dave to provide details

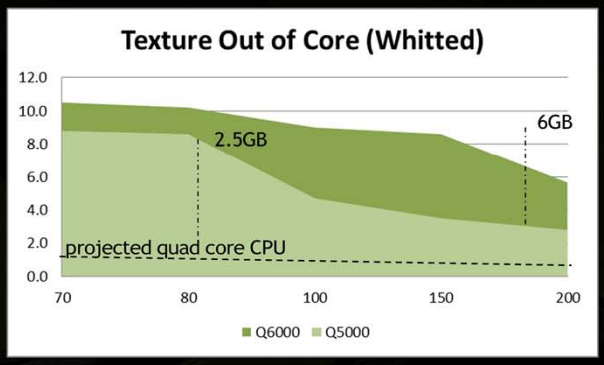


Ray Tracing on Kepler GPUs

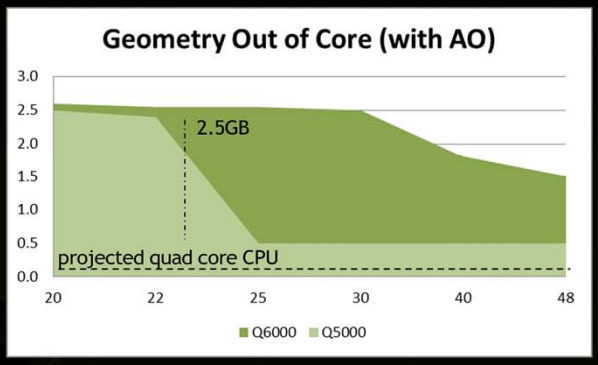
- While OptiX 2.6 and iray 3 now support Kepler GPUs, they are yet to be optimized for the Kepler architecture
- Optimization at NVIDIA will continue as the full-size Kepler GPU (as in the K20) nears availability
- NVIDIA will be sharing all it learns with companies doing their own GPU ray tracing approaches so they can also exploit Kepler
- Be careful in comparing ray tracing performance between recent Kepler-based products and their Fermi predecessors, as this often compares different “size” processors and CUDA core relationships

OptiX 2.5 Out of Core Performance

- Averaged results, as paging impact is view dependent



# of 4k Images

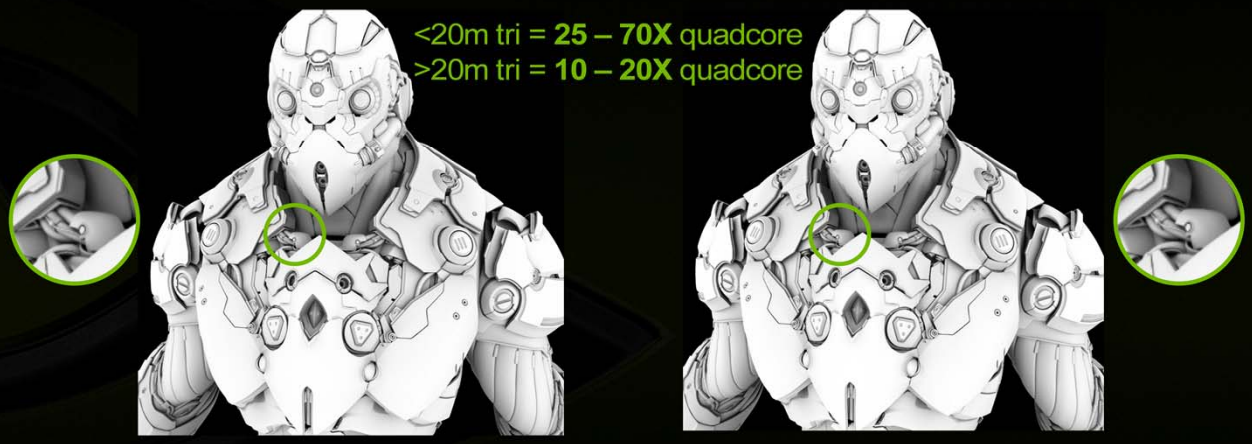


Millions of Textured & Smoothed Faces

Quadro 6000 = 6GB on board memory
Quadro 5000 = 2.5GB on board memory

mental ray Ambient Occlusion

- mental ray 3.11 pipeline accelerated (nearing Beta)



- 1.5sec HLBVH build + 15sec on Quadro 6000 vs. 20 minutes on CPU

OptiX – what's next, and a bitter deeper

- David McAllister
  OptiX Development Manager
  NVIDIA

Introducing OptiX 2.6.

2.6 is about one feature: Kepler support

BTW, all these 3D logos were made with the new ray traced renderer in Adobe After Effects CS6, which is based upon OptiX.

2.6 is about providing one new feature: Kepler support.

It was made from the stable 2.5 code base.

If you are shipping to customers in the very near term, 2.6 is for you.

Not as much ray tracing perf comes for free

With OGL, D3D, and CUDA you see perf increases with each driver rev.

You will see speedups with each OptiX release as well. We have many more optimizations to apply on Kepler and are actively working on that.

Perf per watt improvement

This is an initial implementation; we are hard at work optimizing.

The rest of the perf will come in the K20: more features, better perf

NVVM Optimization

- CUDA 1.0 → 4.0 used Open64 compiler front end
- CUDA 4.1+ uses LLVM
- NVVM = LLVM for CUDA
  - Open source!
- NVVM code generation is very different.
- 2.6 is optimized for NVVM code.

LLVM stands for… It's a …

LLVM is a great leap forward for CUDA. It allows any language to work on the CUDA platform and on OptiX.

We do the dirty work for you

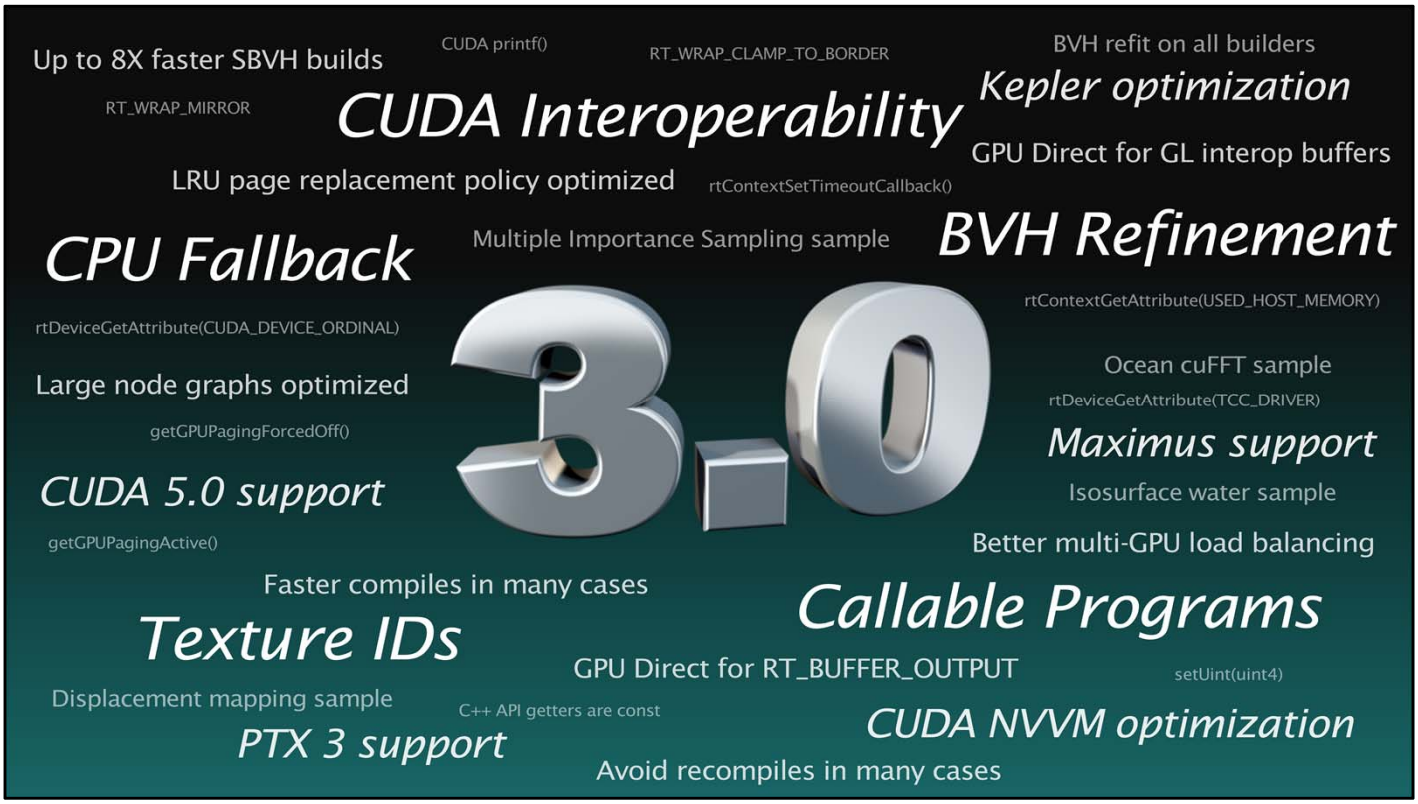
- Kepler Optimization
- NVVM Optimization

It’s like having ten more people on your rendering team.

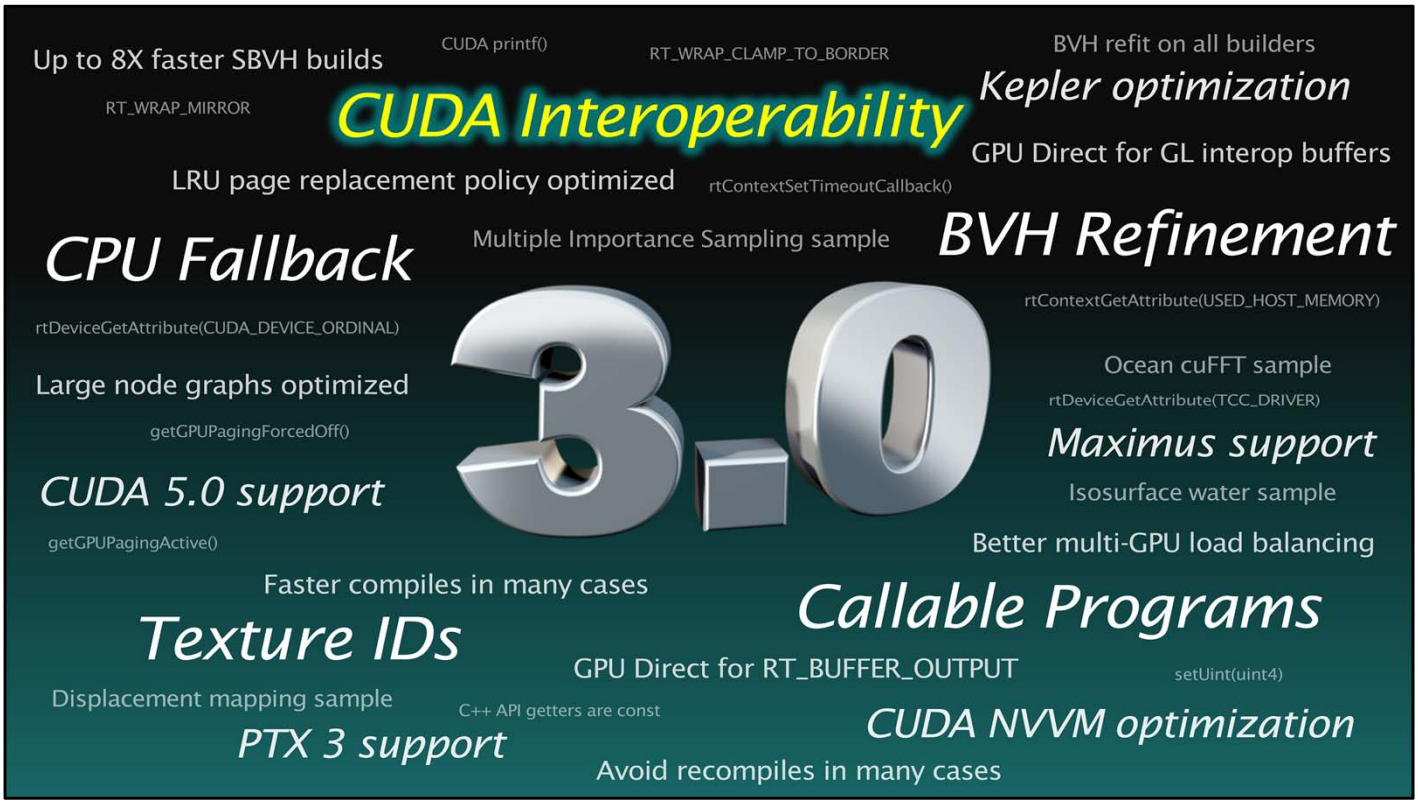
OptiX 2.6 - Available Soon

http://developer.nvidia.com


OptiX 3.0



There are over thirty new features in OptiX 3.0 and I'd like to tell you about each one of them. … but I won't.



Interoperability (AKA “interop”) is the ability to share resources directly with other APIs using the GPU.

OptiX has included Interop Support since day one for OpenGL and Direct3D, so you can share textures and buffers without having to copy the data across the bus and through your application. Many people have asked for OptiX interoperability with CUDA, and we’re pleased to say that 3.0 includes this powerful capability.

OptiX 3.0 – CUDA Interoperability

- Sharing Contexts
- Sharing Pointers
- Multi-GPU

OptiX 3.0 – Sharing CUDA Contexts

- There is a CUcontext on each device.
- CUDA runtime silently manages these.
- OptiX used to create its own CUcontexts.
- Now we share with CUDA:
  - If CUDA runtime has already run we find its CUcontexts.
  - If OptiX runs before CUDA runtime we make new CUcontexts.

The CUDA runtime has a context per device that you use.

OptiX 3.0 - Sharing Pointers with CUDA

- rtBufferSetDevicePointer() - CUDA owns the buffer

```
{
  const float* d_output_probe_buffer;
  cudaSetDevice(0);
  cudaMalloc(&d_output_probe_buffer, moving_obj_count * sizeof float));
  rtBufferSetDevicePointer(buf, optixDevice0, d_output_probe_buffer);
  rtContextLaunch1D(..., moving_obj_count);

  LOS_reduction <<<moving_obj_count, 1>>> (d_moving_objs, d_output_probe_buffer);
}
```

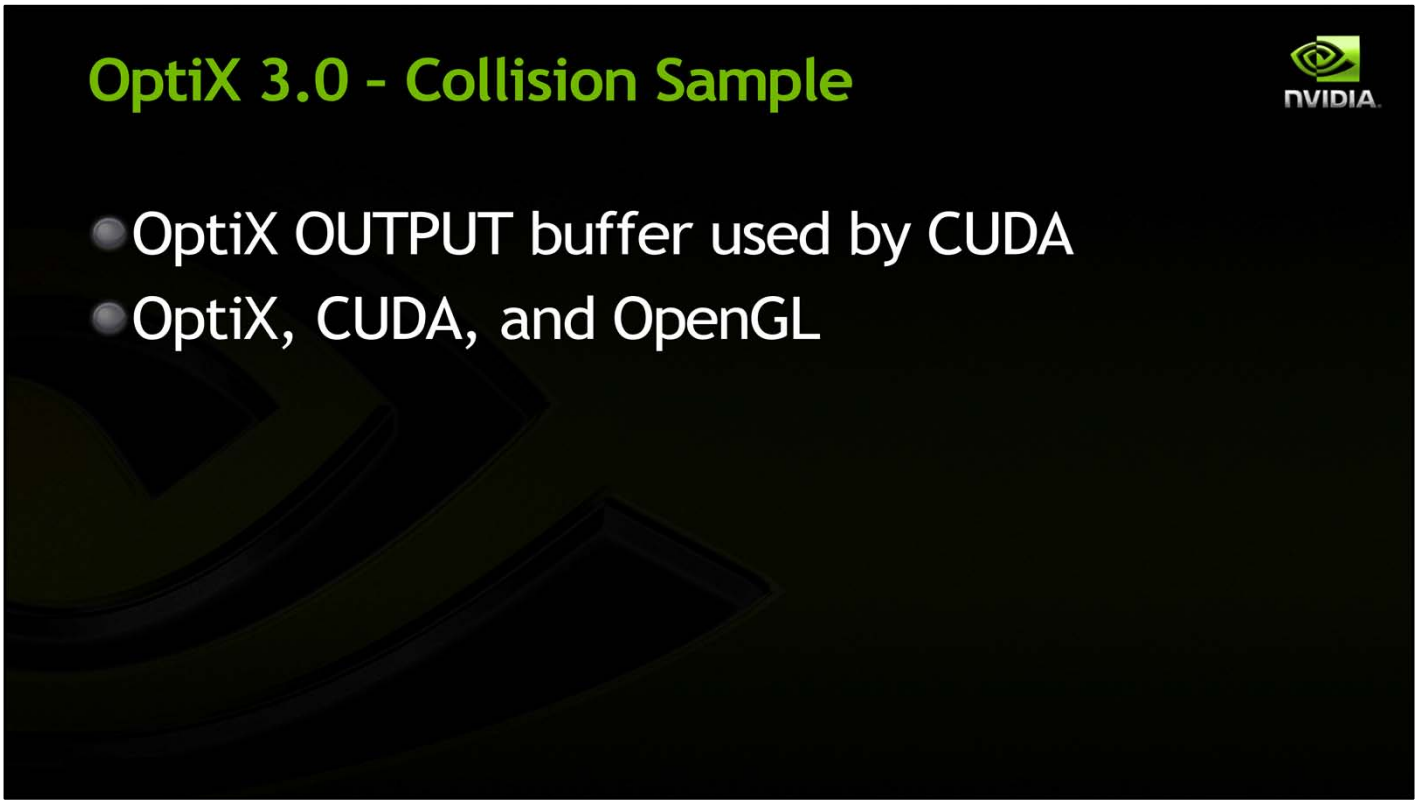
OptiX 3.0 - Sharing Pointers with CUDA

- rtBufferGetDevicePointer() - OptiX owns the buffer

```
{
  rtContextLaunch1D(..., moving_obj_count);
  const float* d_output_probe_buffer;
  rtBufferGetDevicePointer(buf, optixDevice0, &d_output_probe_buffer);

  cudaSetDevice(0);
  LOS_reduction <<<moving_obj_count, 1>>> (d_moving_objs, d_output_probe_buffer);
}
```

As with contexts, we can share buffer data owned by the application, or the application can share buffer data owned by OptiX.

OptiX 3.0 – Collision Sample



Uses OptiX, CUDA, and OpenGL

All pairs line of sight, plus 64 curb feeler rays

Red ones move away from nearest red or green

Greens move away from nearest green but chase nearest red

OptiX 3.0 - Ocean cuFFT Sample

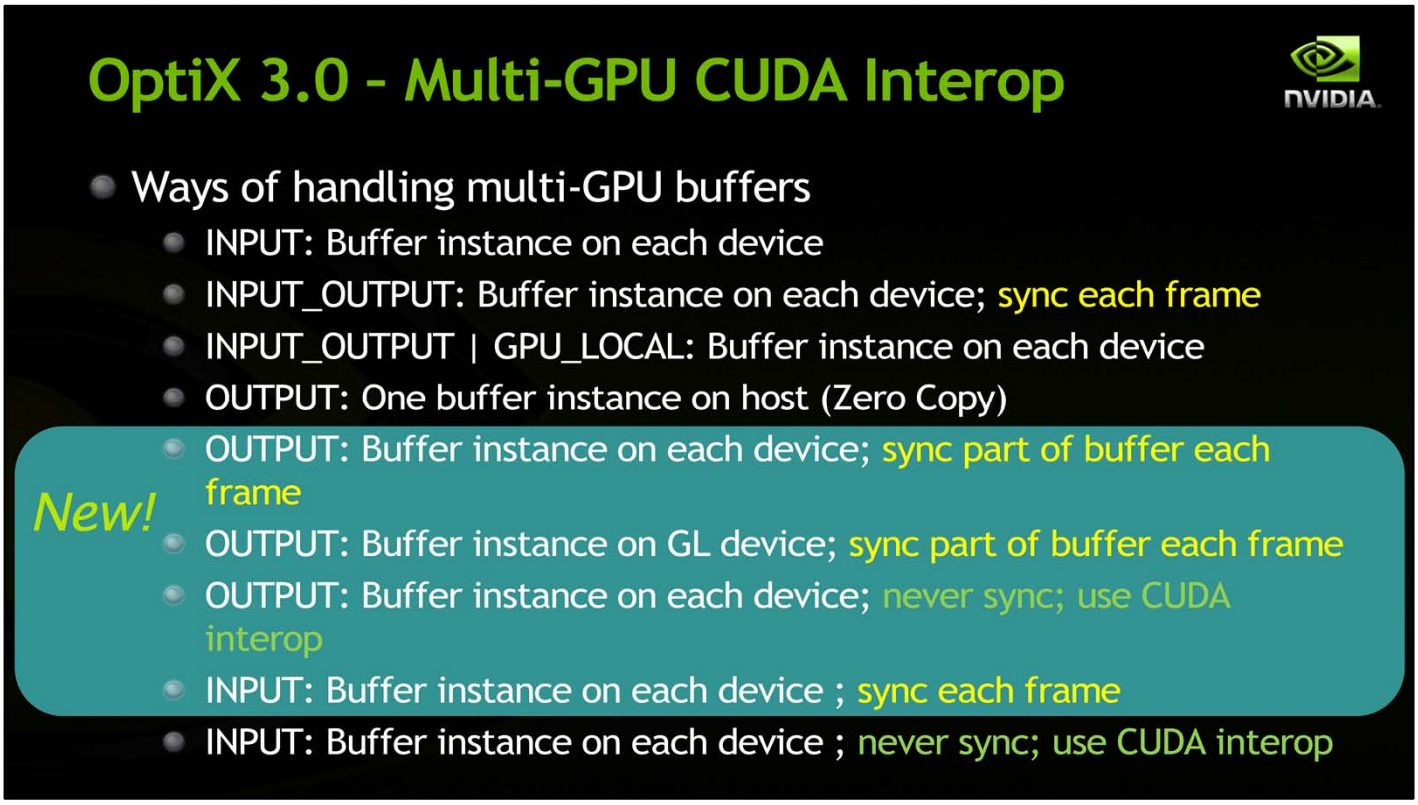
- Tessendorf FFT-based ocean surface algorithm
  - Uses cuFFT
  - 1024x512 simulation
  - 1024x1024 height field primitive
- Water with Fresnel dielectric shading model
  - 6 bounce reflection; 6 bounce refraction
- Preetham physically-based sky model miss program
- CUDA owns the buffer; OptiX uses it as RT_INPUT buffer
- Reinhard tone mapping on RT_OUTPUT buffer

Put Ocean talking points here.

OptiX 3.0 - Water Sample

- PhysX CFD water simulation on one Fermi GPU
  - 128x128x64 volume
- Water ray tracing on two Kepler GPUs
  - Water with Fresnel dielectric shading model
  - 12 bounce reflection; 12 bounce refraction

Put Ocean talking points here.



SAY THIS WHILE OCEAN IS RUNNING

We wanted a way to allow apps to not worry about what devices they are running on.

7.4. Multi-GPU considerations

If the application provides or requests device pointers for all devices on which OptiX is running, no additional data copies need to be made. However, whenever there is a mismatch between the devices on which the application has provided or requested pointers and the devices on which OptiX is running, OptiX will need to make sure that all of its devices have the necessary data.

Note that these issues can arise in some circumstances even when OptiX is only using one GPU. If the application is running CUDA code on one GPU, but has instructed OptiX to only run on another GPU, it is legal to use rtBufferSetDevicePointer to provide a device pointer on the non-OptiX GPU; OptiX will handle any required data transfer internally.

7.4.1. When the application provides pointers to OptiX

If a device pointer is provided for one device but not for all OptiX devices, OptiX will allocate memory on the missing devices and copy the buffer data from the provided pointer to the missing devices during rtContextLaunch. It is a caught runtime error for the application to specify pointers for more than one but less than all devices.

This implementation allows applications to be ignorant, if desired, of whether one or multiple devices are being used for OptiX and whether CUDA is being run on the same or a different device than OptiX. Conversely, the application may be fully in control of which devices run OptiX and which devices run CUDA and fill each device's copy of a buffer either by CUDA or by OptiX.

7.4.2. When the application receives pointers from OptiX

When the application requests a pointer from OptiX (to an RT_BUFFER_INPUT or RT_BUFFER_INPUT_OUTPUT buffer), we assume that the application is modifying the data contained in that buffer. Therefore we keep track of which OptiX devices the application has requested pointers for, and if the application has requested only one pointer but there are additional OptiX devices, we will copy the data from that device to all others on the next launch. If the application requests pointers on all devices, we assume they have set up the data how they want it, and no copying will happen. It is a caught runtime error to request pointers for more than one but fewer than all devices.

Best Practices - Avoid unnecessary copies

- Taking a pointer → OptiX copies its contents to other devices.
- RT_BUFFER_COPY_ON_DIRTY avoids this.

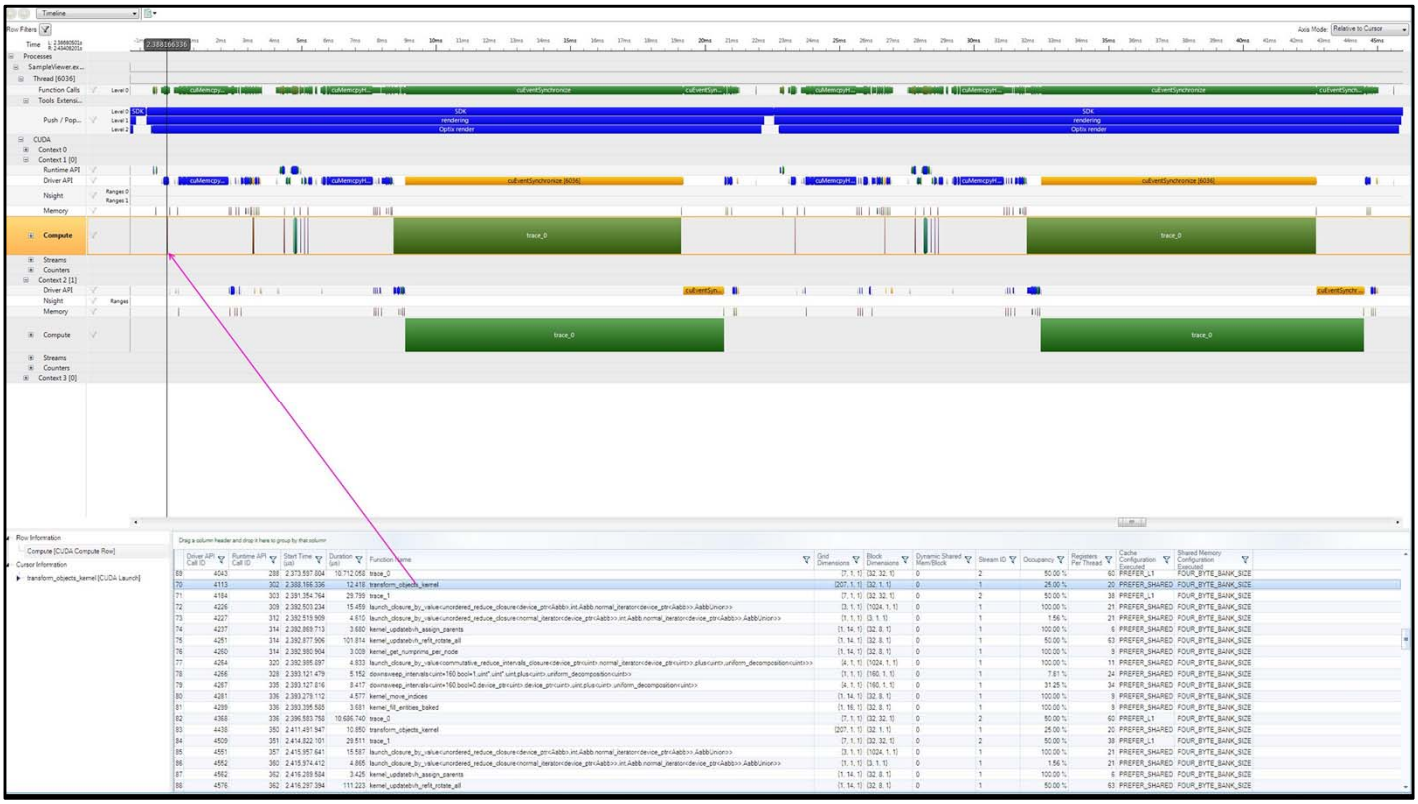
```
{
  RTbuffer buf;
  rtBufferCreateForCUDA(Ctx, RT_BUFFER_INPUT | RT_BUFFER_COPY_ON_DIRTY, &buf);
  // set size and type...
  const float* d_input_buffer;
  rtBufferGetDevicePointer(buf, optixDevice0, d_input_buffer);

  cudaSetDevice(0);
  CoolCUDAKernel<<<count, 1>>>(d_input_buffer);

  rtBufferMarkDirty(buf);
  rtContextLaunch1D(..., moving_obj_count);
}
```

By default any buffer you have a pointer to requires OptiX to copy its contents to other devices.

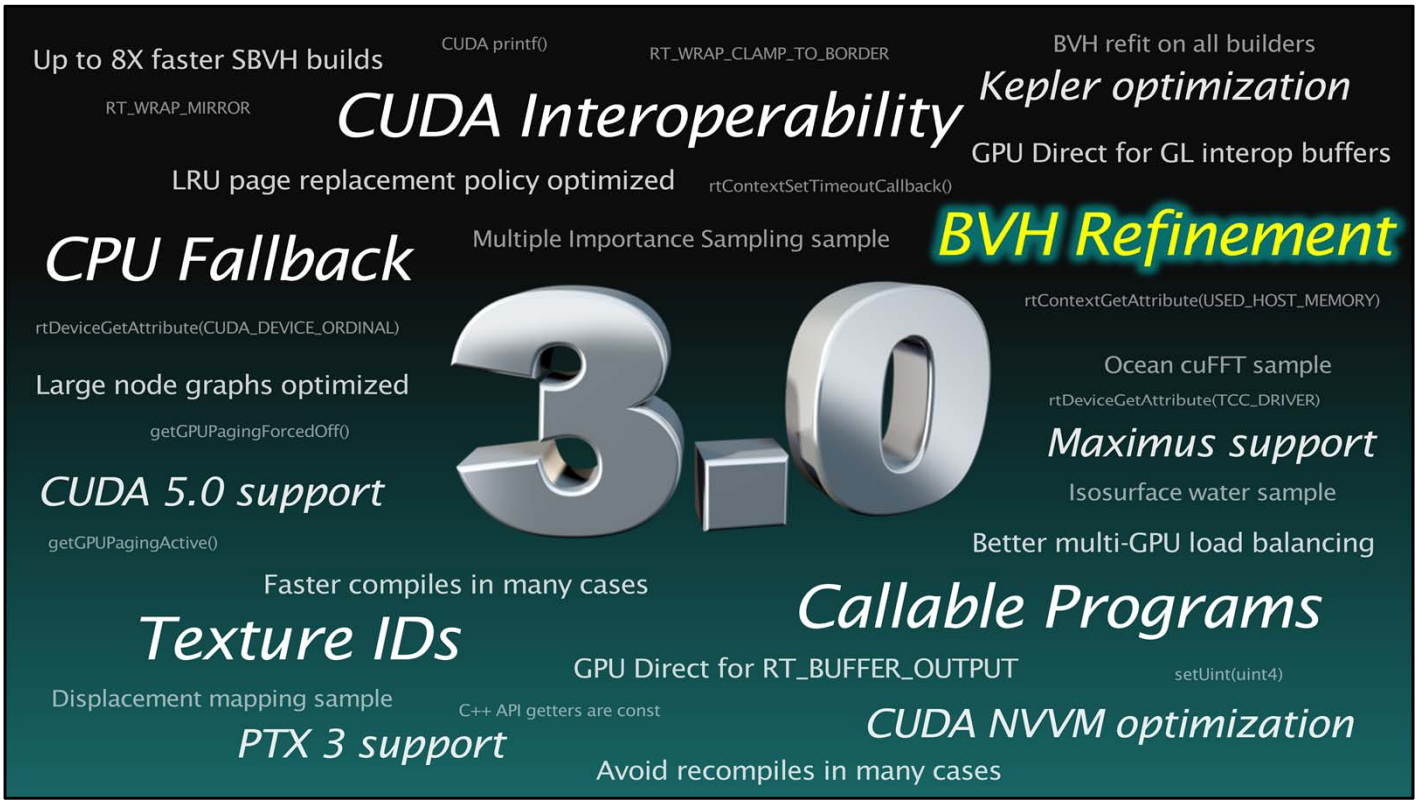
The exception is if you have pointers to the buffer on all devices then we assume you update it yourself.

But let's say you don't update the data every frame. Maybe just on the first frame. Then you don't want OptiX copying it superfluously.

Marking the buffer as COPY_ON_DIRTY means that we won't copy unless you call rtBufferMarkDirty.



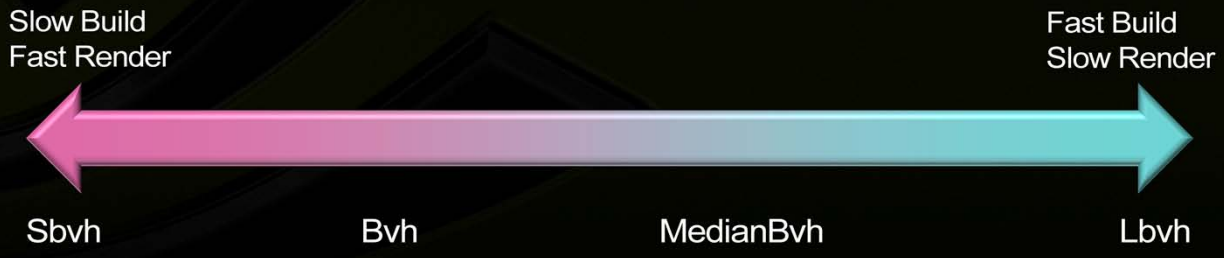
The point: CUDA Interop avoids copies



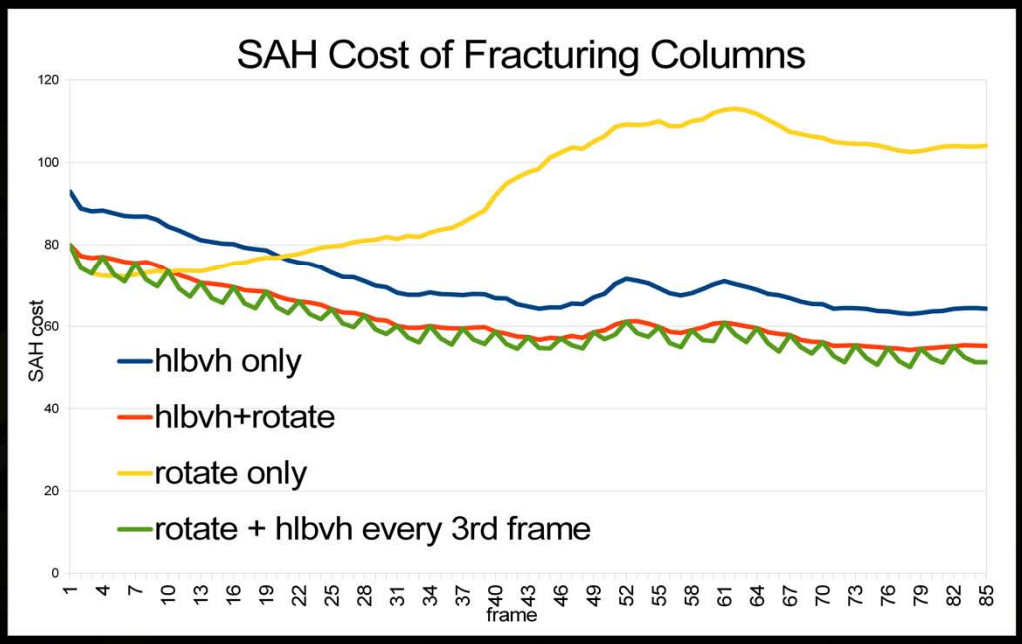
BVH Refinement -

OptiX 3.0 - BVH Refinement

- "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



OptiX 3.0 - BVH Refinement



Lower is better.

OptiX 3.0 - Refit and Refine

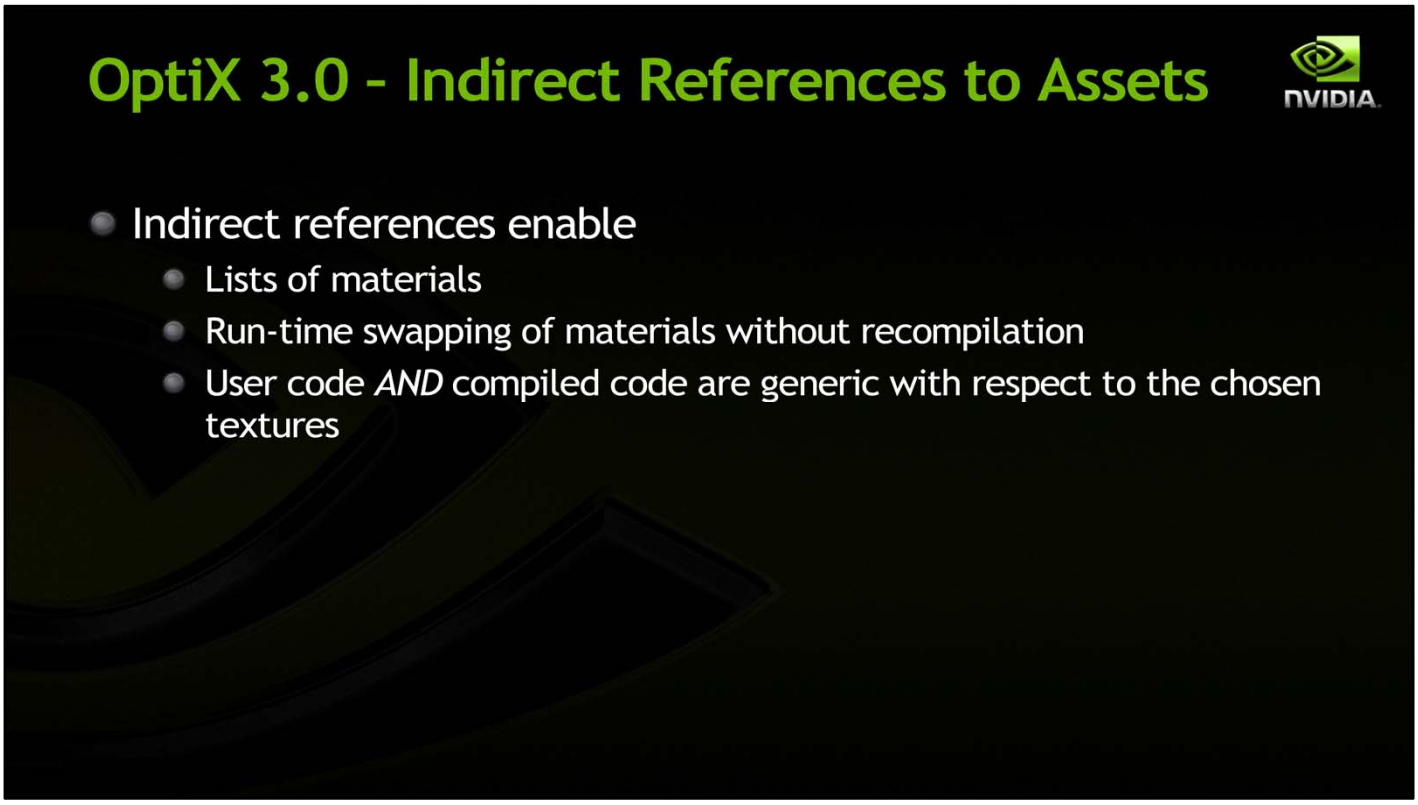
- rtAccelerationSetProperty(accel, “refit”, “1”)
  - 0→rebuild whenever dirty
  - 1→refit every frame if prim count constant; else rebuild
  - >1→ refit every frame; refine every N^{th} frame if prim count constant; else rebuild
- rtAccelerationSetProperty(accel, “refine”, “8”)
  - 0→rebuild or refit; never refine
  - 1→ refit and refine once per frame if prim count constant; else rebuild
  - >1 →refit and refine N times per frame if prim count constant; else rebuild
- Both work on all BVH builders.

OptiX 3.0 - Fracture Demo

- BVH Refinement: “refit”=1 “refine”=8
- NVIDIA PhysX GPU Rigid Bodies
- CUDA Interop for geometry
- OpenGL Interop for TXAA
- Glass shader with Fresnel reflection
- About 350,000 triangles
- Max ray depth of 12

BVH Refinement with One refine per frame; One rebuild per eight frames
NVIDIA PhysX GPU Rigid Bodies
CUDA – OptiX Interop for geometry
OpenGL – OptiX Interop for TXAA
Glass shader with Fresnel reflection
About 350,000 triangles
Max ray depth of 12

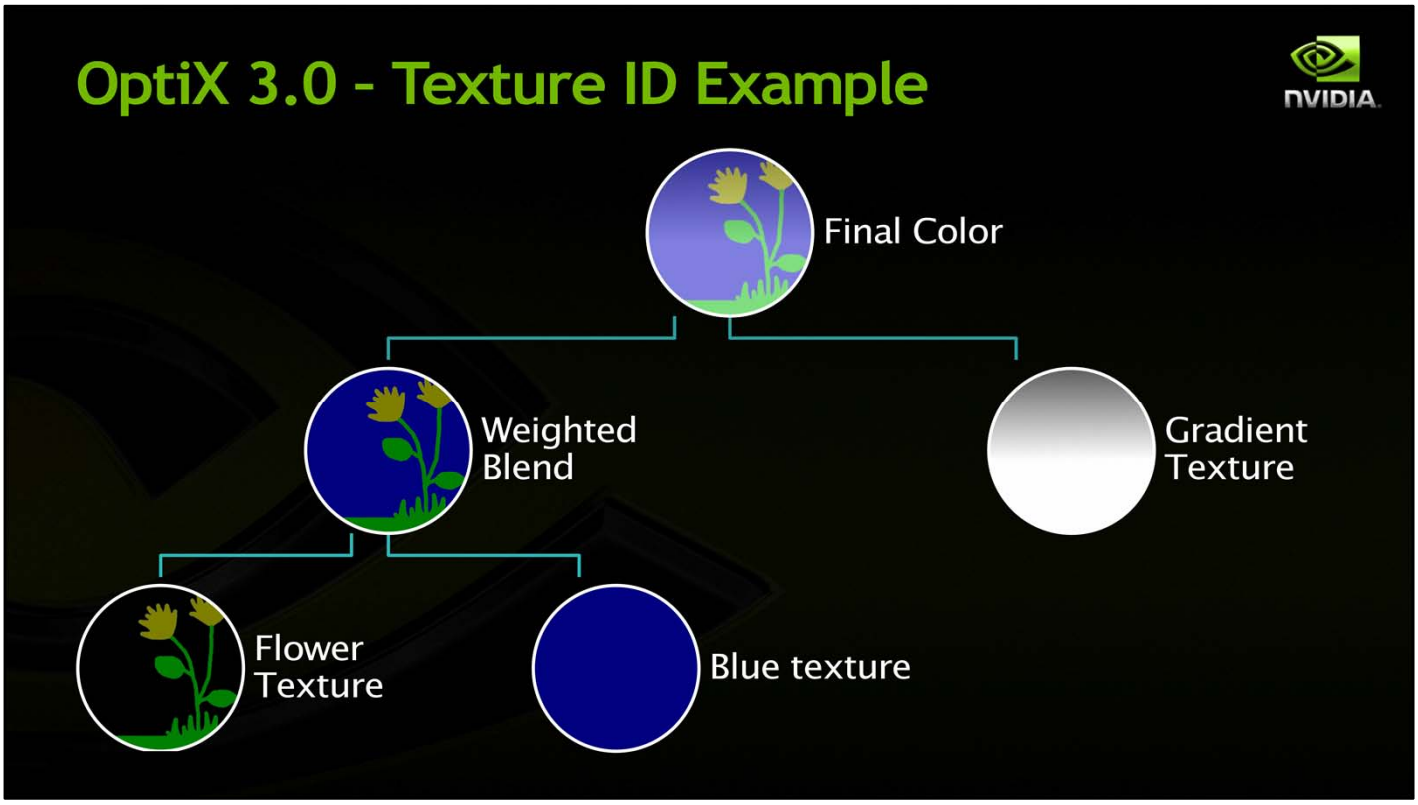
Texture ID Support



As part of our effort to extend the OptiX programming model to be more generic…

We wanted a way to provide indirect texture access.

OptiX 3.0 - Texture ID Example

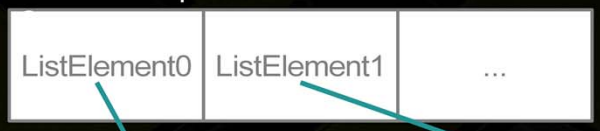


Graph Layout


```
struct Bitmap {
    int bindless_id;  // -1 unused
    int texture_type; // 0-bitmap, 1-blend
}
```

```
struct ListElement {
    int index_in_texture_buffer;
    float weight;
}
```

List of Bitmaps for Diffuse

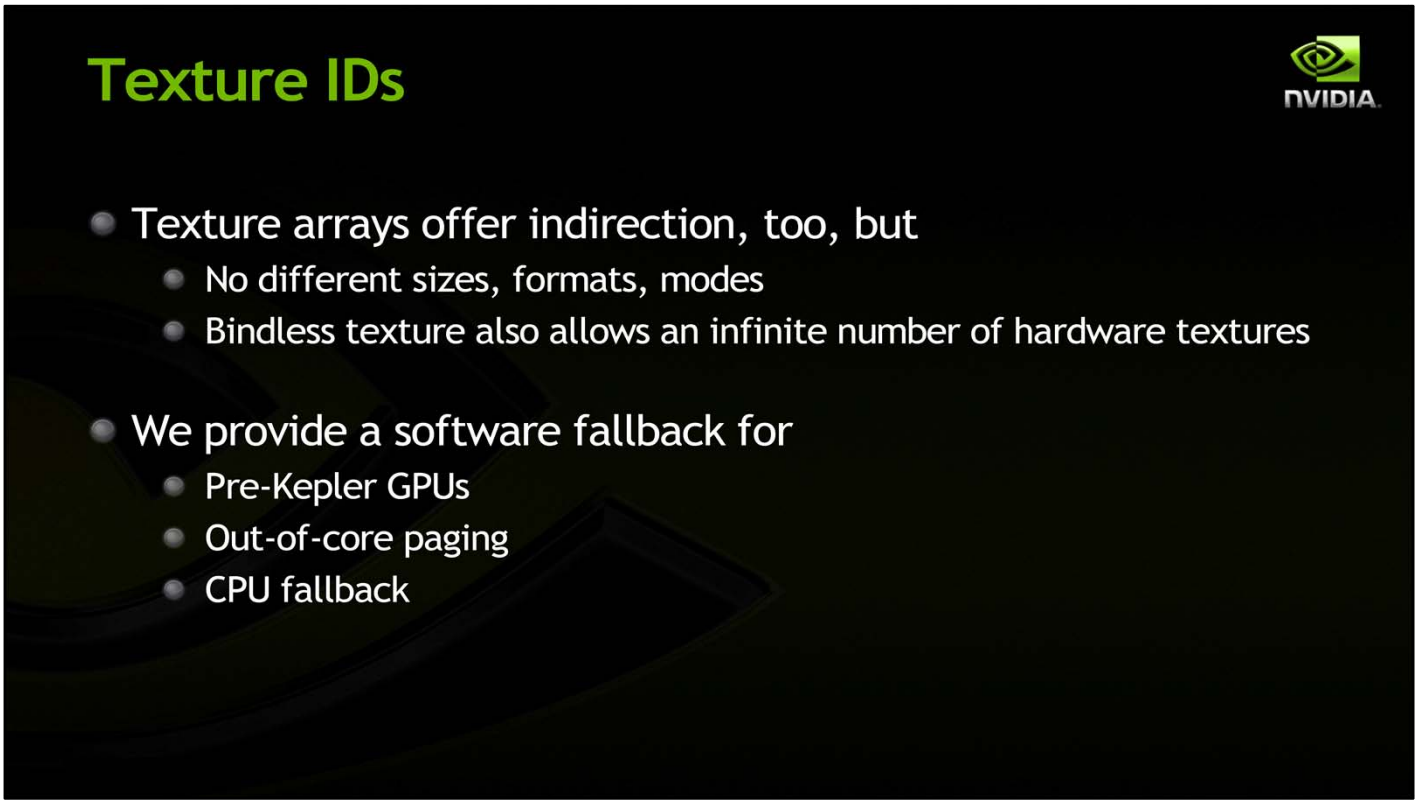


Buffer with All Bitmap Information

Kepler Bindless Texture

- OLD:
  - .tex .u64 diffuse_map;
  - tex.2d.v4.f32.f32 {%f6,%f7,%f8,%f9},[diffuse_map,{%f1,%f2,%f3,%f4}];
- NEW:
  - .reg .u64 diffuse_ind;
  - tex.2d.v4.f32.f32 {%f6,%f7,%f8,%f9},[%diffuse_in {%f1,%f2,%f3,%f4}];

I'll show you how this is manifested in Kepler PTX and then in OptiX



Bindless means indirect. Indirect means flexible programming

Bindless helps to avoid OptiX and OCG recompiles

OptiX bindless is more flexible than texture arrays (can have different sizes and modes)

rtTextureId is int and rtTextureSampler's lifetime identifier

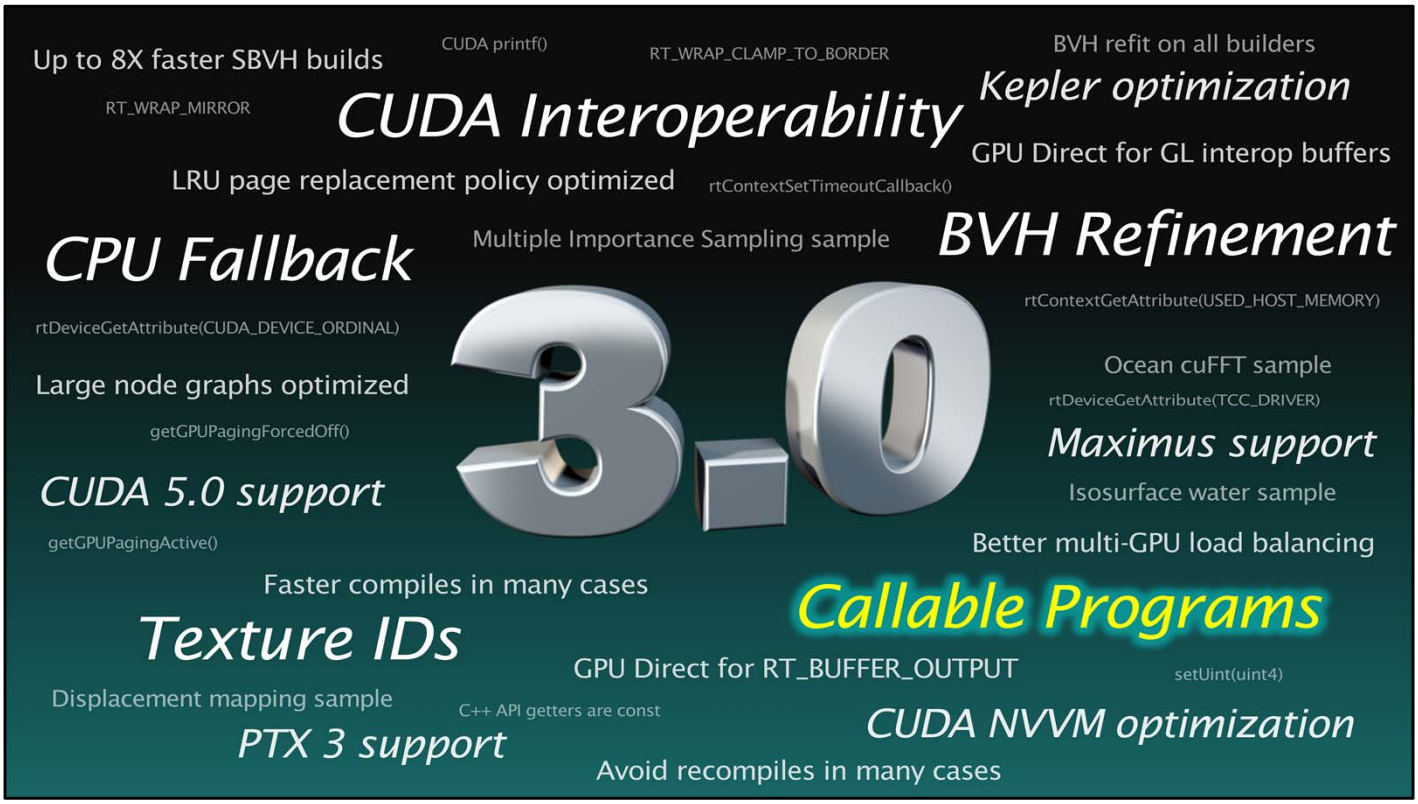
rtTextureId is independent of underlying SW/HW implementation

HW bindless on Kepler should be as fast as a switch statement of direct textures. We are working on this

SW bindless fallback is 2-3X slower than SW fallback of direct texture (CPU, paging, pre-Kepler)

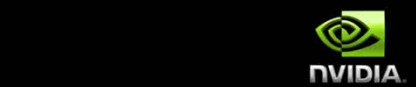
HW bindless texture on Kepler is unlimited number of textures too

A new clamping modes are just to match OpenGL/CUDA.



Callable Programs – making shade trees possible

Callable Programs



- Enables shade trees
- Selectable filtering, noise functions, gamma functions, etc.

Very highly requested capab

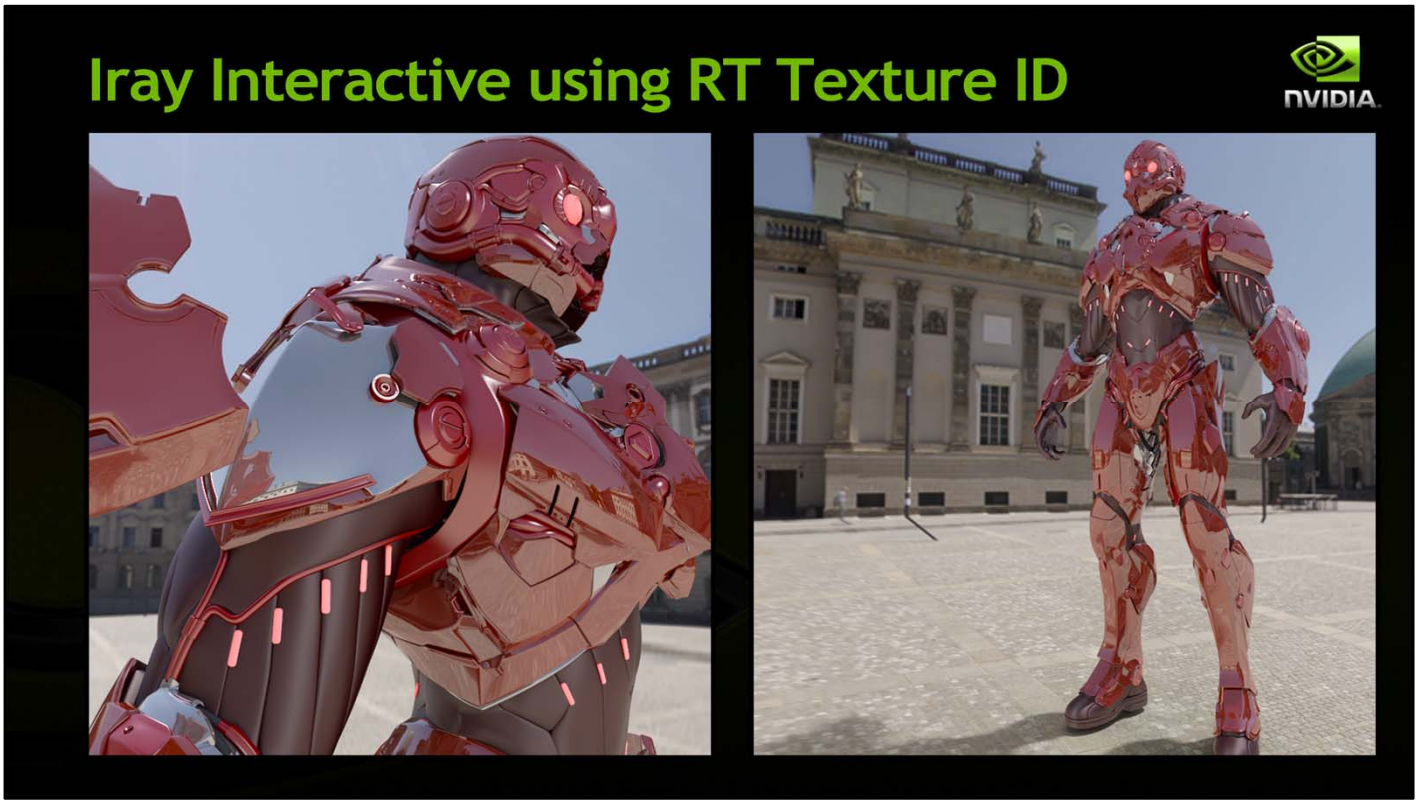
Callable Programs - Device


```
RT_CALLABLE_PROGRAM float3 checker_color(float3 input_color, float scale)
{
  uint2 tile_size = make_uint2(launch_dim.x / N, launch_dim.y / N);
  if (launch_index.x/tile_size.x ^ launch_index.y/tile_size.y)
    return input_color * scale;
  else
    return input_color;
}
rtCallableProgram(float3, get_color, (float3, float));
RT_PROGRAM camera()
{
  float3 initial_color;
  // … trace a ray, get the initial color …
  float3 final_color = get_color( initial_color, 0.5f );
  // … write new final color to output buffer …
}
```

1) Note the float3 return type and RT_CALLABLE_PROGRAM

Callable Programs - Host

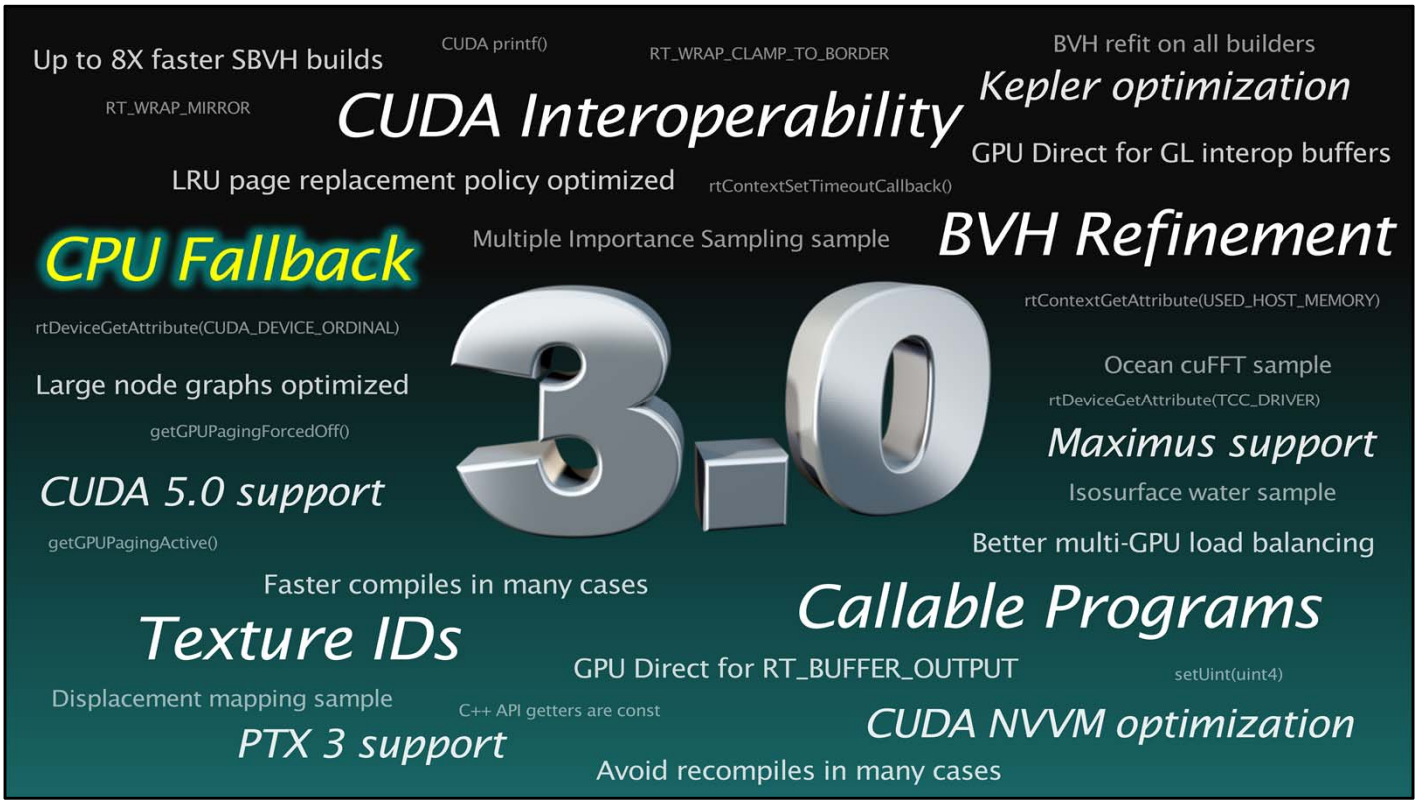

```
RTprogram color_program;
RTvariable color_program_variable;

rtProgramCreateFromPTXFile( context, ptx_path, “get_color”, &color_program );
rtProgramDeclareVariable( camera_program, “get_color”, &color_program_variable );
rtVariableSetObject( color_program_variable, color_program );
```



Heiko’s notes

Greg’s example

Marc’s shots



CPU Fallback: This is the ability for your OptiX-based application to render on the CPU when you don't have an Nvidia GPU.

OptiX 3.0 - CPU Fallback

- Write once
  - Useful for new applications
  - One code path to maintain
- Run anywhere (no NVIDIA driver required)
  - Windows, Linux, Mac
  - NVIDIA, AMD, Intel
- Available for applications with large installed bases
  - Example: Adobe After Effects CS6
  - Companies can apply via: optix-help@nvidia.com

NVIDIA has shipped a lot of CUDA-capable, and thus OptiX-capable GPUs.
Why would we want to provide a CPU fallback?
The more broadly adopted the application, the more general the hardware support needs to be – it's also much more difficult for us to support.
As a result, we offer the CPU fallback feature only to select companies on a contractual basis who serve a large and diverse user base.

CPU Fallback - In Use

- Same executable with no changes
  - Run on NVIDIA hardware when present
  - Run on CPU when NVIDIA hardware or driver absent
- Application can choose CPU if desired

In the near term – use OptiX 2.6 for production/shipping applications and the 3.0 Beta to explore new capabilities and give us feedback – we read it ALL!


NVIDIA
For Your Information
setenv CUDA_VISIBLE_DEVICES 0,1,2
Turn SLI off!

You can mail us traces so we can reproduce bugs AND so we can optimize OptiX for your use case

Last time we had OptiX talks by Lego, Audio, Adobe, and CCP Games. Next time we would love to have a talk by you.

uniformly regarded as an essential resource for scientists, developers, graphic artists, designers, researchers, engineers, and IT managers, who rely on GPUs to tackle enormous computational challenges.

There are three ways you can participate…

(1) Speak - share your work and gain exposure as a leader in the visualization community.

(2) Register to attend and learn from the experts and network with your peers. Exclusive to Siggraph attendees is a special 10% discount off the full conference rate. Use promo code **GM10SIGG** through March 17, 2013. Registration for GTC 2013 will open in late November/early December.

(3) Sponsor and Exhibit – GTC attracts influential decision-makers from a broad range of industry verticals, so this is a great event to reach people who manage significant IT budgets.