Release Notes for the NVIDIA® OptiX™ ray tracing engine

Version 3.0.1 August 2013

Welcome to the latest release of the NVIDIA OptiX ray tracing engine and SDK, with support for all CUDA-capable GPUs. This package contains the libraries required to experience the latest technology for programmable GPU ray tracing, plus pre-compiled samples (with source code) demonstrating a broad range of ray tracing techniques and highlighting basic functionality.

Support:

The normal path for OptiX support is on NVIDIA's Developer Zone at: https://devtalk.nvidia.com/default/board/90/

If you have any confidential concerns please send your issues directly to OptiX-Help@NVIDIA.com and they will be addressed by the development team. You can continue to download OptiX from http://developer.nvidia.com/optix-download/

System Requirements (for running binaries referencing OptiX)

Graphics Hardware:

- CUDA capable devices (G80 or later) are supported on **GeForce**, **Quadro**, or **Tesla** class products. Kepler GK104, GK107 and GK110 GPUs are now supported. Multiple devices/GPUs are only supported on "GT200", "Fermi" and "Kepler" generation GPUs. Out-of-core ray tracing of large datasets exceeding GPU memory (AKA paging) is not supported on GeForce GT class GPUs.

Graphics Driver:

- The CUDA R300 or later driver is **required**. The latest drivers available are highly recommended (307.4 or later for Windows, 310.19 for Linux and the CUDA 5.0 driver extension for Mac). For the Mac, the driver extension module supplied with CUDA 5.0 or later will need to be installed. Note that the Linux and Mac drivers can only be obtained from the CUDA 5.0 download page at the moment.
- SLI is not required for OptiX to use multiple GPUs, and it interferes when OptiX uses either D3D or OpenGL interop. Disabling SLI will not degrade OptiX performance and will provide a more stable environment for OptiX applications to run. SLI is termed "Multi-GPU mode" in recent NVIDIA Control Panels, with the correct option being "Disable multi-GPU mode" to enable OptiX to freely use all system GPUs.

Operating System:

- Windows XP/Vista/7 32-bit or 64-bit; Linux RHEL 4.8 - 64-bit only, Ubuntu 10.10 - 64-bit; OSX 10.6+ (universal binary with 32 and 64-bit x86).

Development Environment Requirements (for compiling with OptiX)

All Platforms (Windows, Linux, Mac OSX):

- **CUDA Toolkit 2.3, 3.0, 3.1, 3.2, 4.0, 4.1, 4.2, 5.0.**
  OptiX 3.0.1 has been built with CUDA 4.2 and 5.0, but any specified toolkit should work when compiling PTX for OptiX. If an application links against both the OptiX library and the CUDA runtime on Mac and Linux, it is recommended to use the same version of CUDA as OptiX was built against. Note that CUDA 5.5 is not supported in this release.
- **C/C++ Compiler**
  Visual Studio 2005, 2008 or 2010 is required on Windows systems (Visual Studio 2012 is not supported with this release). gcc 4.4-4.6 have been tested on Linux. The 4.5 Xcode development tools have been tested on Mac OSX 10.8.
- **GLUT**
  Most OptiX samples use the GLUT toolkit. Freeglut ships with the Windows OptiX distribution. GLUT is installed by default on Mac OSX. A GLUT installation is required to build samples on Linux.

Enhancements in OptiX 3.0.1

- Optimized all kernel launch times by approximately two ms.
- Optimizations for certain recompilation cases.
- Fixed ocean and collision samples to help them compile in the released SDK.
- Fixed liboptix's Mac rpath to pick up the cudart lib which is installed alongside liboptix because cudart is not binary compatible between versions.
- Constrained exported symbols on Mac and Linux to just the OptiX API calls. Relinking might be necessary.
- Fixed bugs with changing the set of devices being used and changing between GPUs of different compute capabilities.
- Fixed bug with propagating global/const initializers when inlining.
- Fixed out of memory errors with paging.
- Fixed bug with material indices being clear to zero when using the clustered triangles primiticve.
- Mark Acceleration dirty when rtAccelerationSetData fails.

Enhancements since OptiX 3.0.0 Beta 1 release

- **Callable Programs**
  Enhanced callable programs to now inherit the semantic type from the caller. If you call a callable program from a closest hit program, you can do all the same things as you could with closest hit programs such as access attribute variables and call rtTrace.

  In addition, the lookup scopes for callable programs has changed. The lookup scope now mimics that of the caller. For example a callable program called from an any_hit program will look up values from the callable program itself, GeometryInstance, Material, and finally the Context as seen from the caller.

  Support for passing pointers to local stack variables to callable programs for sm_1x devices. See programming guide for more information.

  Fixes for various types of parameter passing.
- Fixed a bug that prevented creating more than a single OptiX context in a given process.

- Improved CUDA context handling. If you had a context set before calling OptiX, that context is restored. If there was no context set, the context from the first OptiX device will be current.
- Various compile time improvements.
- Various fixes for Kepler based GPUs.
- Various runtime optimizations for Kepler GPUs, especially Tesla K20.
- Fix --machine/-m keywords in rtuCUDACompile()
- Fix for crashes related to unattached texture samplers without associated buffers.
- CUDA's printf should work in addition to OptiX's rtPrintf. Note that CUDA's printf does not respect any of OptiX's printf APIs such as rtContextSetPrintBufferSize or rtContextSetPrintLaunchIndex.
- Utilizing 32-bit PTX within a 64-bit application has been disabled in this version.

Enhancements since OptiX 2.6 final release:

- **Callable Programs**
  Enables the assignment of arbitrary CUDA functions to OptiX variables, enabling a more diverse set of shading and rendering algorithms such as shade trees. See the OptiX 3.0 Programming Guide and API Reference for documentation.
- **OptiX-CUDA Interop**
  Enables efficient sharing of buffers between OptiX and CUDA. CUDA contexts are now shared between OptiX and the CUDA runtime. RT_BUFFERS may have their storage set to a preexisting CUDA buffer. Pointers to RT_BUFFER contents may be queried to allow CUDA to read and write contents of OptiX buffers. See OptiX Programming Guide chapter 7 and API Reference for documentation.
- **Texture Identifiers and CUDA Bindless Textures**
  It is now possible in the host API to query the CUDA texture handle of an RT_TEXTURE and to store this handle in OptiX variables and buffers, thereby allowing indirect access to textures. This enhances the programming model, enabling shade trees and more generic code. See OptiX Programming Guide and API Reference for documentation.
- **TCC Driver Support**
  OptiX now works as expected when Tesla cards use the TCC driver for either single or multiple GPU usage, allowing OptiX applications to be fully compatible with NVIDIA Maximus system configurations.
- **New SDK Samples**:
  - **callablePrograms** – a basic example of callable programs
  - **mis_sample** – shows multiple importance sampling
  - **rayDifferential** – shows texture identifiers as well as ray differentials for texture filtering
  - **ocean** – shows CUDA interop by modeling ocean waves in cuFFT and rendering in OptiX
- **BVH Refinement**
  A BVH is only valid while the geometry is static and must be rebuilt if the geometry deforms or changes topology. The original "refit" property of the acceleration structure mitigated this if the number of primitives stayed the same and the deformation was minimal. The new "refine" property allows the BVH to be refit under more extreme deformation and actually improve. Refinement can also be applied to static geometry to cause the BVH to improve beyond the quality of the original builder. This makes the fast, lower quality builders (such as LBVH) more usable for more situations.
- **Much Faster SBVH builds**
  Performance for SBVH has improved by approximately 800%

- **Better Multi-GPU Load Balancing**
  The algorithm for partitioning work across multiple GPUs has improved, increasing multi-GPU scalability.
- **Improved Performance via GPU Direct for GL Interop Buffers**
  GPU Direct (available on Fermi and Kepler professional GPUs) is now supported so that OptiX buffers can be copied between GPUs without copies through the host. GL interoperability otherwise only worked within a single GPU (and still does on Consumer GPUs).
- **RT_WRAP_MIRROR** and **RT_WRAP_CLAMP_TO_BORDER**
  These texture wrap modes have been added to match the corresponding new CUDA functionality.
- **rtContextSetTimeoutCallback**
  The application may provide a function pointer to a function that is called back from long-running OptiX host API calls, such as rtContextCompile and rtContextLaunch. The callback function can ask OptiX to terminate the long-running call. This helps interactive applications avoid slow response times.

Known limitations with version 3.0.1:

- There is a known performance decrease of 5-10% when running on drivers 304.00 through 306.00 on Windows, 304.00 through 304.43 on Linux, CUDA library version 5.0.0 through 5.0.28 on Mac. This slow down is fixed in 307 and later drivers.
- Using multiple GPUs with mixed TCC/WDDM modes (on Windows) is only supported on R304 or later drivers.
- Utilizing 32-bit PTX within a 64-bit application has been deprecated.
- Support for building host-based acceleration structures in parallel has been disabled on Linux in this version of OptiX.
- OptiX supports running with NVIDIA Parallel Nsight but does not currently support kernel debugging in Nsight. In addition, it is not recommended to compile PTX code using any -G (debug) flags to nvcc.
- Performance of OpenGL and DirectX interop is unpredictable when SLI is enabled. As noted previously, SLI is not required to achieve scaling across multiple GPUs and is not recommended when using OptiX.
- All GPUs used by OptiX must be of the same MAJOR compute capability, such as compute capability 1.x or 2.x. OptiX will automatically select the set of GPUs of the highest major compute capability and only use those. For example, in a system with a GeForce GTX 460 (compute 2.1) and a GeForce GTX 480 (compute 2.0), both will be used, but in a system with a Quadro 5800 (compute 1.3) and a Quadro 6000 (compute 2.0) only the compute 2.0 device would be selected. One exception of this is compute 3.5 devices (e.g. K20 and K20x) cannot be mixed with compute 3.0 devices (e.g. QK5000 and K10). Applications may explicitly choose which GPUs to run, as is done in the progressive photon mapper sample, ppm.cpp, at the start of initScene(), but if the application requests a set of devices of different major compute capability an error will be returned.
- Texture arrays and MIP maps are not yet implemented. Texture identifiers are more general than texture arrays.
- malloc() and free() do not work in device code.
- Applications that use RT_BUFFER_INPUT_OUTPUT or RT_BUFFER_OUTPUT buffers on multi-GPU contexts must take care to ensure that the stride of memory accesses to that buffer is compatible with the PCIe bus payload size. Using a buffer of type RT_FORMAT_FLOAT3, for example, will cause a massive slowdown; use RT_FORMAT_FLOAT4 instead. Likewise, a group of parallel threads should present a contiguous span of 64 bytes for writing at once on an Intel chipset to avoid massive slowdowns, or 16 bytes on NVIDIA chipsets to avoid moderate slowdowns.
- Linux only: due to a bug in GLUT on many Linux distributions, the SDK samples will not restore the original window size correctly after returning from full-screen mode. A newer version of freeglut may avoid this limitation.

- The CUDA release notes recommend the use of -malign-double with GCC. However, on Mac OSX systems (10.5 with GCC 4.0.1 and 4.2.1 and 10.6 with GCC 4.2.1) this flag can produce miscompiles with std::stream based classes in host code when compiling to 32 bits. If the structs are different sizes between device and host code, consider manually padding the structure rather than using this compiler flag.
- Several OptiX SDK samples are intended to run on a single GPU and will perform more slowly when run on multiple GPUs. The samples: Cook, Hybrid Shadows, isgReflections, isgShadows, and SimpleAnimation will only run at full speed on a single GPU. The compiled versions of these samples can be constrained to a single GPU by setting the environment variable CUDA_VISIBLE_DEVICES = N, where "N" is the number of the GPU you wish to utilize. The OptiX sample3 is a convenient means to quickly identify your device ID's.

Performance Notes:

- OptiX performance tracks very closely to a GPU's CUDA core count and core clock speed for a given GPU generation. The performance "value" of a CUDA core varies greatly between GPU generations (e.g., GT200, Fermi, Kepler) but can be compared directly within the same GPU generation.
- OptiX is continuing to be tuned for Kepler, with the goal to increase performance on Kepler based GPUs in a future release.
- Different techniques may find varying performance results on different GPU generations. For example, path tracing benefited more from the Fermi architecture than did simpler techniques as compared to how they performed on GT200.
- OptiX takes advantage of multiple GPUs without using SLI. Multi-GPU scalability will vary with the workload being done, with longer and complex rendering (e.g., path tracing) scaling quite well while simple and fast rendering (e.g. Whitted) may scale far less.
- Mixing board types will reduce the memory size available to OptiX to that of the smallest GPU.
- Performance will be better when the entire scene fits within a single GPU's memory. Adding additional GPUs increases performance, but does not increase the available memory beyond that of the smallest board. The entire scene must fit within GPU memory when paging is disabled or not available.
- For compute-intensive rendering, performance is currently fairly linear in the number of pixels displayed/rendered. Reducing resolution can make development on entry level boards or laptops more practical.
- Performance on Windows Vista and 7 will be somewhat slower than Windows XP or Linux due to the architecture of the Windows Display Driver Model (WDDM).
- Uninitialized variables can increase register pressure and negatively impact performance.
- Pass arguments by reference instead of value whenever possible when calling local functions for optimal performance.

Other Notes:

- **CMake 2.8.6** (at least 2.6.3; 2.8.10.2 also works.) http://www.cmake.org/cmake/resources/software.html The executable installer http://www.cmake.org/files/v2.8/cmake-2.8.6-win32-x86.exe is recommended for Windows systems.