Release notes for the NVIDIA® OptiX™ ray tracing engine

Version 2.1.1 June 2010

Welcome to the next release of the NVIDIA OptiX ray tracing engine and SDK, with support for all CUDA-capable GPUs. Within this package you will find the libraries required to experience the latest technology for programmable GPU ray tracing, plus pre-compiled samples (with source code) demonstrating classic ray tracing results and other basic functionality.

Support:

Please post comments or support questions on the NVIDIA developer forum that can be found here: http://developer.nvidia.com/forums/index.php?showforum=43. Questions that require confidentiality can be e-mailed to OptiX-Help@NVIDIA.com and someone on the development team will get back to you, although perhaps not as quickly as via the forum.

System Requirements (for running binaries referencing OptiX)

Graphics Hardware:

- CUDA capable devices (G80 or later) are supported equally on either **GeForce**, **Quadro**, or **Tesla** class products. Multiple devices/GPUs are only supported on "GT200" or "Fermi" class GPUs.

Graphics Driver:

- The CUDA R260 or later driver is **required**. The latest drivers available are highly recommended (260.03 or later for Windows, 260.19.21 for Linux and the CUDA 3.2 driver extension for Mac). For the Mac, you will need to install the driver extension module supplied with CUDA 3.2 or later.

Operating System:

- Windows XP/Vista/7 32-bit or 64-bit, Linux 32-bit or 64-bit, OSX 10.5+ (universal binary with 32 and 64 bit x86).

Development Environment Requirements (for compiling OptiX)

All Platforms (Windows, Linux, Mac OSX):

- **CUDA Toolkit 2.3, 3.0, 3.1, 3.2, or 4.0.**
  OptiX 2.1 has been tested with CUDA versions 2.3, 3.0, 3.1, 3.2, and 4.0.
- **CMake 2.6.4** (at least 2.6.3; 2.8.4 is the current version and also works.)
  http://www.cmake.org/cmake/resources/software.html
  The executable installer http://www.cmake.org/files/v2.6/cmake-2.6.4-win32-x86.exe is recommended for Windows systems.
- **C/C++ Compiler**
  Visual Studio 2005 or 2008 is required on Windows systems. gcc 4.2 and 4.3 have been tested

on Linux. The 3.1 and 3.2 Xcode development tools have been tested on Mac OSX 10.5 and 10.6.

Performance Notes:

- OptiX performance tracks very closely to a GPU's CUDA core count and core clock speed for a given GPU generation.
- CUDA applications, such as those based on OptiX, take advantage of multiple GPUs without using SLI. We recommend not configuring your GPUs in SLI mode for OptiX applications.
- Mixing board types will reduce the memory size available to OptiX to that of the smallest GPU.
- The entire scene must fit within a single GPU's memory to be processed by OptiX. Adding additional GPUs increases performance, but does not increase the available memory beyond that of the smallest board.
- 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 practical.
- Performance on Windows Vista and 7 may be somewhat slower than Windows XP due to the architecture of the Windows Display Driver Model.
- Uninitialized variables can increase register pressure and negatively impact performance.
- Pass arguments by reference instead of value whenever possible when calling local functions.

Changes from version 2.1.0

- Fixed known issues when targeting sm_20.
- Fixed several bugs related to using Transforms in user CUDA programs.
- Fixed some compilation issues when using double precision in user CUDA programs.
- Fixed optixpp_namespace.h compile error when using SelectorObj::getChild(), GroupObj::getChild(), and TransformObj::getChild().

Changes from version 2.0.0 Highlights

- Support for PTX generated by **CUDA 3.1**, **CUDA 3.2** and **CUDA 4.0**.
- Support for **64 bit PTX**.
- New **rtuTraverse API** for tracing batches of rays against geometry with CPU and GPU support.
- A standard **`intersect_triangle()`** function is provided in `optixu_math_namespace.h`.
- A new set of header files that encapsulate all C++ classes and API into an **'`optix`' namespace**.

Changes from version 2.1.0 Release Candidate 2

- Fix compiles against CUDA 4.0. PTX generated by CUDA 4.0 has not been fully tested and is not recommended for production use at this time. Specifying -arch sm_20 to nvcc will generate code that is known to be incompatible with OptiX; sm_10 targeted PTX generally works, but has not been fully tested.
- Fixed (removed) cerr printing in rtuTraverse calls.

- Fixed bug where querying the amount of free memory with `rtContextGetAttribute` before `rtContextLaunch` caused an error on 64 bit systems.
- Work around nvcc having problems with UNC paths in `rtuCUDACompileFile`.
- Fixed bug with Median Bvh on 64 bit systems.
- Fix for bug with Whirligig when using the /fp:fast flag on for Visual Studio.

Changes from version 2.1.0 Release Candidate 1

- RtuTraverse API changes
  - Sped up CPU fallback path
  - See documentation for new capabilities, including optional return data for ray hits
  - Since flags have change, apps must be recompiled against RC2.
  - Added traversal sample
  - Doesn't link against `nvcuda.dll` or `nvcuda.so`. This allows apps linking to OptiX to execute on machines that do not have an NVIDIA driver installed. This also enables the the CPU fallback option of rtuTraverse.
- Sped up SBVH builds and some other OptiX code
- Replaced `inline` keyword in optix_math with `__forceinline__` to speed up both Windows and GCC host code. This can be overridden by redefining `OPTIXU_INLINE` before `#including optix_math.h`.
- Removed all instances of `.f64` from PTX files by forcing floating point literals to float instead of double in SDK samples.
- Changed Visual Studio optimization flags for release builds of SDK, including `/fp:fast`
- Inverted sign of normal returned by `intersect_triangle` in `optixu_math_namespace.h` (it was previously returning `-normal` instead of `normal`).
- Fixed bug on G80 GPUs where it complained about `cuMemAlloc`
- Fixed bug with SwimmingShark sample on OBJ models that included more than one group
- Fixed bug regarding inlining certain function calls in SM2.0 PTX.
- Fixed warning: comma at end of enumerator list
- -malign-double is not used anywhere.

Changes from version 2.0.0

- Support for PTX generated by **CUDA 3.1, CUDA 3.2** and **CUDA 4.0 B2.**
  Note that malloc, free, printf and function pointers are not currently supported by OptiX.
- Support for **64 bit PTX**
  Optix 2.1 now supports 64 bit user PTX, allowing the full 6GB of memory on Quadro 6000 and Tesla C2070 boards to be accessed.
  - The NVIDIA driver release 260 or newer is required to execute 64 bit PTX. Using an older driver with 64-bit code will result in a runtime error.
  - Attempt to use 64 bit PTX on a 32 bit host application will result in a runtime error.
  - All user programs must have the same pointer size (32 or 64 bits); attempting to mix and

match 32 and 64 bit PTX will result in a runtime error.

  - In order to use `rtuCUDACompileFile` to generate 64 bit PTX, you must explicitly include `-m64` in the compilation flags.

  - For performance reasons, users are encouraged to continue to use 32 bit PTX even in a 64 bit host application unless they have a need for the additional address space.

- Performance improvement for most heavy ray tracing workloads. Rendering intensive applications may experience a speedup of up to 80%. **However, slowdowns are also possible.** Typically, the slowdowns occur on scenes with very few primitives, such as the simpler SDK samples. Performance of small scenes on SM 1.0 chips (G80-based, GeForce 8800 GTX, for example) receive a larger slow down than newer GPUs.

- New sample: MCMC - Markov Chain Monte Carlo method rendering. A global illumination solution that requires an SM 2.0 class device (e.g. Fermi) or higher.

- New **API**: rtuTraverse
  This API provides a wrapper around OptiX for calculating the nearest intersection for a batch of rays and a list of triangles. No additional OptiX API calls are required, making it straightforward to integrate into existing applications that simply need to trace rays without shading, such as light baking and collision detection. rtuTraverse additionally includes a CPU implementation. See `include/optixu/optixu_traversal.h` and the OptiX_Utility_Library_Reference documentation for details.

- New documentation for OptiXpp C++ interface in a Doxygen generated file (see the OptiX_Utility_Library_Reference document) and in Chapter 7 of the Programming Guide.

- A standard `intersect_triangle()` function is provided in `optixu_math_namespace.h`. This simplifies implementation of triangle intersection programs and ensures that applications have the best-in-class implementation.

- A new set of header files have been added that are designed to encapsulate our C++ based classes and API as well as CUDA's vector types into a namespace called '`optix`'. In the `optixu` include directory are several new files that when included don't pollute the global namespace with OptiX functions or types (e.g. CUDA's `float3`). Backward compatibility is maintained if you include the old headers. It is not recommended to mix the old global namespace versions of the headers with the new optix namespace versions of the headers. Doing so can result in Linker errors and type confusion.

  - `optixu_math_namespace.h`
  - `optixupp_namespace.h` (backward compatibility with `optixu::` namespace is provided in `optixpp.h`.)
  - `optixu_matrix_namespace.h`
  - `optixu_aabb_namespace.h`
  - `optixu_math_stream_namespace.h`
  - `optixu_vector_types.h`
  - `optixu_vector_functions.h`

  In addition, there is also a new top-level header file that is designed to offer all that OptiX provides via the C and C++ API as well as additional helper classes. This header is `<optix_world.h>` and can be included in C or C++ code as well as CUDA C code. Note that this header includes the namespace versions of the headers (namespace `optix`). Future additional helper classes or functions will be added to this header.

- Added support for `__float2half()` and `__half2float()` CUDA intrinsic functions.

Known limitations with version 2.1.1:

- The LBVH acceleration structure is not functional in this release; if a user attempts to use it, it will be internally replaced by the MedianBVH acceleration structure.
- The internal format for Acceleration Structure data has changed going from 2.0 to 2.1.0. Previous cached data will not be usable with 2.1 and must be regenerated.
- OptiX currently does not support running with NVIDIA Parallel Nsight. In addition it is not recommended to compile PTX code using any -g (debug) flags to nvcc.
- Use of OpenGL and DX interop causes OptiX to crash when SLI is enabled. As previously noted, SLI is not required to achieve scaling across multiple GPUs.
- All GPUs used by OptiX must be of the same compute capability, such as compute capability 1.3 or 2.0. OptiX will automatically select the set of GPUs of the highest 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), the compute 2.1 device would only be selected. 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 compute capability an error will be returned.
- There currently is a concurrent texture limit of 127 textures. This limit will be entirely removed in the near future, although large numbers of textures will always be likely to negatively impact performance. An error is returned by OptiX if this limit is exceeded.
- Texture arrays and MIP maps are not yet implemented.
- malloc, free, and printf 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.
- Under certain circumstances it is possible that OpenGL buffer / OptiX interop is not working properly. An upcoming display driver update, starting with version 270.00, will resolve this issue. In particular cases the following sequence of commands will fail:
    - Create an OpenGL buffer and frame buffer object
    - Create an OpenGL texture
    - Attach the OpenGL texture to the frame buffer object (GL_COLOR_ATTACHMENT0)
    - Call glReadPixels to copy from GL_COLOR_ATTACHMENT0 to the OpenGL buffer
    - Create an OptiX buffer from the OpenGL buffer
    - Launching your OptiX kernel might fail now

  When you are affected by this behavior you can try to use the OpenGL texture directly in OptiX and skip the copy operation to the OpenGL buffer via glReadPixels; e.g. as shown in the isgShadows sample.
- 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 your structs

are different sizes between device and host code, consider manually padding the structure rather than using this compiler flag.