NVIDIA® OptiX™ Ray Tracing SDK

Release Notes

Version 7.2.0 **October 2020**

Welcome to the 7.2.0 release of the OptiX SDK. This release includes support for module specializations for runtime feature toggles, AOV denoising for increased flexibility, improved error checking and debugging for an enhanced developer experience, as well as source libraries and sample code to support demand loading of rendering data.

Upgrading to 7.2.0 may require source code changes, though they should be minimal. Applications compiled with the 7.2.0 SDK headers will require driver version 455 or later. Applications compiled with earlier SDK headers will continue to work, but recompiling with 7.2.0 may improve performance and will expose new features.

System Requirements (for running binaries referencing OptiX)

Graphics Hardware:

- All NVIDIA GPUs of Compute Capability 5.0 (Maxwell) or higher are supported.

Graphics Driver:

- OptiX 7.2.0 requires that you install a r455+ driver.
- Windows 8.1/10 64-bit; Linux RHEL 4.8+ or Ubuntu 10.10+ 64-bit

CUDA Toolkit

- It is **not** required to have any CUDA toolkit installed to be able to run OptiX-based applications.

Development Environment Requirements (for compiling with OptiX)

- CUDA Toolkit 7, 8, 9, 10, 11, 11.1

  Any specified toolkit should work when compiling PTX for OptiX, but the latest version is recommended. OptiX is compatible with both the CUDA device and runtime APIs.

  Version 11.1 of the CUDA toolkit introduces CUDA sparse textures used in OptiX's new demand loading library. This library is built only when the OptiX SDK is compiled with CUDA 11.1.

- C/C++ Compiler

  A compiler compatible with the used CUDA Toolkit is required. Please see the CUDA Toolkit documentation for more information on supported compilers.

What's New in this version

- Specialization is a powerful new feature that allows renderers to maintain generality while increasing performance on specific use cases. A single version of the PTX can be supplied to OptiX and specialized to toggle specific features on and off. The OptiX compiler is leveraged to fold constant values and elide complex code that is not required by a particular scene setup. Specialized values are supplied during module creation with `OptixModuleCompileOptions::boundValues`. See the Programming Guide section 6.3.1, "Parameter specialization", and the `optixBoundValues` sample.
- Demand loading source library
  - Enables textures to be loaded on demand, which greatly reduces memory requirements, start-up time, and disk I/O compared to preloading textures.
  - Requires the CUDA 11.1 toolkit.
  - See section 14 "Demand-loaded sparse textures" of the OptiX Programming Guide for a detailed technical introduction.
  - The `optixDemandTexture` sample demonstrates how to use this library.
  - This library supersedes the lower-level `optixPaging` library. The `optixDemandPaging` sample shows how to use `optixPaging` directly.
  - Known issue: wrap mode and mirror mode are not yet supported for CUDA sparse textures.
- There is a new mode in the denoiser that uses a neural network to predict a filter kernel instead of the final image. The filter weights can be applied to multiple layers or AOVs with just an incremental cost.
  - To select this mode, use `OptixDenoiserModelKind::OPTIX_DENOISER_MODEL_KIND_AOV`
  - Additional quality may be achieved by computing the average color using a new API function `optixDenoiserComputeAverageColor` and supplying the value to the denoiser using `OptixDenoiserParams::hdrAverageColor`
- To aid in debugging there is a new validation mode that runs additional checks during runtime and enables all debug exceptions.
  - Enable validation mode with: `OptixDeviceContextOptions::validationMode`
  - New error code when validation catches an error: `OPTIX_ERROR_VALIDATION_FAILURE`
  - APIs that take `CUstream` arguments are synchronized on the stream to check for errors before proceeding.
  - optixLaunch will synchronize after the launch and report errors
  - If any OptiX debug exceptions were thrown during launch, a CUDA launch error is triggered to prevent proceeding.
  - Validation mode reduces performance. Remember to turn it off.
- New debug exceptions
  - `OPTIX_EXCEPTION_CODE_CALLABLE_INVALID_SBT`
    - The callable program SBT record index was out of bounds

    - `OPTIX_EXCEPTION_CODE_CALLABLE_NO_DC_SBT_RECORD`
        - The callable program SBT record does not contain a direct callable program
    - `OPTIX_EXCEPTION_CODE_CALLABLE_NO_CC_SBT_RECORD`
        - The callable program SBT record does not contain a continuation callable program
- When linking, unresolved and multiply defined symbols will produce detailed error messages in the logs.
- Instance bounds are now computed automatically, even with motion and motion transforms. Instance acceleration structure build inputs no longer have the optional `OptixBuildInputInstanceArray::aabbs`.
    - Note OptiX will compute AABBs for all applications running against driver 455+ regardless of the version of the SDK it was built against. OptiX will ignore the AABBs supplied by applications compiled with earlier SDKs.
- You can now unload the NVIDIA OptiX driver DLL or DSO. See `optixUninitWithHandle()` in the `optix_stubs.h` header file.
    - If other threads in the process have a handle to the DLL it will continue to be loaded until the last handle is released.
    - There is a new error code when `optixUninitWithHandle` fails: `OPTIX_ERROR_LIBRARY_UNLOAD_FAILURE`
- Fixed a bug where PTX compiled with -lineinfo in CUDA 11 could cause errors when loading into OptiX.
- Fixed bugs related to large numbers of curves.
- Optimized traversal when only triangle geometry is enabled with single level instancing, that is, `OptixPipelineCompileOptions::usesPrimitiveTypeFlags` equals exactly `OPTIX_PRIMITIVE_TYPE_FLAGS_TRIANGLE` and `OptixPipelineCompileOptions::traversableGraphFlags` equals exactly `OPTIX_TRAVERSABLE_GRAPH_FLAG_ALLOW_SINGLE_LEVEL_INSTANCING`
- SDK
    - By default the SDK will target PTX compilation for SM 60 (Volta+). Set CMake variables `CUDA_NVCC_FLAGS` and `CUDA_NVRTC_FLAGS` to use an older SM version if desired.
    - New samples
        - optixBoundValues
        - optixDemandTexture
        - optixDenoiser
        - optixDynamicGeometry

Known Issues

1. Pixel formats OPTIX_PIXEL_FORMAT_UCHAR3 and OPTIX_PIXEL_FORMAT_UCHAR4 are not supported by the Denoiser.
2. Concurrent launches from the same pipeline will serialize automatically on the device.
3. OPTIX_COMPILE_DEBUG_LEVEL_FULL does not currently generate debug information necessary for cuda-gdb or Nsight Compute VSE.

What's New in 7.1.0

- Added curves as a new type of geometric primitive. Curves are swept surfaces used to represent long thin strands, such as for hair, fur, or cloth fibers. Linear, quadratic, and cubic B-spline bases are supported. Motion blur is supported.
    - Added new GAS build input type for curves. See the OptiX Programming Guide section 5.2, “Curve build inputs”.
    - Hit programs can access the curve parameter value at the hit point, and the curve’s geometric data which is stored in the acceleration structure (GAS). Utility code is provided in the SDK (cuda/curve.h) to compute the curve surface position, tangent, and normal.
    - Each SBT program group for curves requires a built-in curves intersection program, returned by the new host function `optixBuiltinISModuleGet`. Motion blur for curves is enabled here.
    - PIpelines can now indicate which primitive types they support via `OptixPipelineCompileOptions::usesPrimitiveTypeFlags`. If your scene contains curves, they must be enabled here. If your scene geometry is all triangles (no curves and no custom primitives), set these flags to enable only triangles, for optimal performance.
    - In hit programs, the recommended method to discriminate among hit types is to use the new methods `optixGetPrimitiveType`, `optixIsFrontFaceHit`, and `optixIsBackFaceHit`. The old methods, e.g. `optixIsTriangleFrontFaceHit`, still work.
    - See the OptiX Programming Guide section 8, “Curves”, for additional information.
- The denoiser has several improvements for quality and performance in addition to some API changes.
    - Added support for `OptixDenoiserInputKind::OPTIX_DENOISER_INPUT_RGB_ALBEDO_NORMAL`.
    - Added support for tiling. See optix_denoiser_tiling.h in the SDK for helpers to use tiling.
    - `OptixDenoiserOptions::pixelFormat` has been removed, because it is no longer needed.
    - `OptixDenoiserSizes::minimumScratchSizeInBytes` and `recommendedScratchSizeInBytes` have been removed and replaced with `withOverlapScratchSizeInBytes` and `withoutOverlapScratchSizeInBytes`.
- Increase instancing limits. Query limit with `OPTIX_DEVICE_PROPERTY_LIMIT_MAX_INSTANCES_PER_IAS`. The limit has been increased to 2^28 instead of 2^24. `OptixInstance::sbtOffset` now also supports values up to 2^28.
- Removed `OptixPipelineLinkOptions::overrideUsesMotionBlur`. This option is no longer needed.
- Added `OptixTransformFormat` used in OptixBuildInputTriangleArray::transformFormat. Allows to specify the format of `OptixBuildInputTriangleArray::preTransform`. Value should be `OPTIX_TRANSFORM_FORMAT_MATRIX_FLOAT12` when preTransform will be supplied to the build and `OPTIX_TRANSFORM_FORMAT_NONE` when preTransform is unused. A nullptr can now be used for `optixAccelComputeMemoryUsage` instead of supplying a valid or dummy pointer.

- Several new device exceptions were added to catch common errors. They are active when debug exceptions are enabled in `OptixPipelineCompileOptions::exceptionFlags`.
    - Invalid ray exception
        - Checks for NaN and Inf in the ray parameters passed optixTrace
        - New exception code: `OPTIX_EXCEPTION_CODE_INVALID_RAY`
        - `optixGetExceptionInvalidRay` returns details of the exception in a struct called OptixInvalidRayExceptionDetails
    - Callable program parameter mismatch
        - Currently only checks the number of parameters and not the types, since PTX can lose type information.
        - New exception code: `OPTIX_EXCEPTION_CODE_CALLABLE_PARAMETER_MISMATCH`
        - optixGetExceptionParameterMismatch returns details of the exception in a struct called OptixParameterMismatchExceptionDetails
    - Ensure that the built-in intersection program assigned to the SBT matches the GAS
        - New exception code: `OPTIX_EXCEPTION_CODE_BUILTIN_IS_MISMATCH`
    - Ensure that when optixTrace is called with a single level GAS as the trace target that `OptixPipelineCompileOptions::traversableGraphFlags` is set to either `OPTIX_TRAVERSABLE_GRAPH_FLAG_ALLOW_SINGLE_GAS` or `OPTIX_TRAVERSABLE_GRAPH_FLAG_ALLOW_ANY`.
        - new exception code: `OPTIX_EXCEPTION_CODE_UNSUPPORTED_SINGLE_LEVEL_GAS`
    - The shader encountered an unsupported primitive type when calling `optixGetLinearCurveVertexData`, `optixGetQuadraticBSplineVertexData`, `optixGetCubicBSplineVertexData`, or `optixGetPrimitiveType`. Supported primitive types are set with `OptixPipelineCompileOptions::usesPrimitiveTypeFlags`.
        - New exception code: `OPTIX_EXCEPTION_CODE_UNSUPPORTED_PRIMITIVE_TYPE`
- When `OPTIX_EXCEPTION_CODE_TRAVERSAL_INVALID_HIT_SBT` is thrown, `optixGetPrimitiveIndex` is no longer supported. `optixGetSbtGASIndex` should be used instead. See `optixDumpExceptionDetails` for details.
- `optixThrowException` is now elided when user exceptions are disabled instead of producing an error.
- Added `optixGetExceptionLineInfo` device function accessible in exception programs.
    - Returns a string that includes information about the source location that caused the current exception. Only supported for certain exceptions, and requires line information in the PTX (`--lineinfo`) and a debug level that supports line info (`OPTIX_COMPILE_DEBUG_LEVEL_LINEINFO` and `OPTIX_COMPILE_DEBUG_LEVEL_FULL`).
- Change `OptixCompileOptimizationLevel` and `OptixCompileDebugLevel` enum values.

    - Added a DEFAULT value of 0 (used when zero initializing the structs).
    - `OptixCompileOptimizationLevel::OPTIX_COMPILE_OPTIMIZATION_DEFAULT` continues to mean `OPTIMIZATION_LEVEL_3`.
    - `OptixCompileDebugLevel::OPTIX_COMPILE_DEBUG_LEVEL_DEFAULT` (new) adds lineinfo (same as `OPTIX_COMPILE_DEBUG_LEVEL_LINEINFO`).
- Fixed support in optix headers for cuda runtime compilation using nvrtc.
- Enable compaction support for acceleration structures on non-RTX GPUs.
- OptiX will attempt to reset a corrupted compile disk cache.
- Added `OptixIndicesFormat::OPTIX_INDICES_FORMAT_NONE` which must be used when vertex indices are not present (e.g. triangle soup).
- Added `OptixVertexFormat::OPTIX_VERTEX_FORMAT_NONE` for use when initializing `OptixBuildInputTriangleArray::vertexFormat`. The value, `vertexFormat`, must be set to something other than `OPTIX_VERTEX_FORMAT_NONE` before `OptixBuildInputTriangleArray` can be used.
- Additional checks are now in place when calling `optixPipelineSetStackSize`. If the `maxTraversableGraphDepth` is greater than maximum supported by the device an error is generated. `maxTraversableGraphDepth` also must be greater than zero. `maxTraversableGraphDepth` must also be compatible with `OptixPipelineCompileOptions::traversableGraphFlags`.
- When used with Nsight Compute, `optixLaunch` and `optixAccelBuild` are now marked in the timeline.
- Very large AABBs are now clamped to +/- 2^40 for non-motion acceleration structures.
- `optixGetTriangleVertexData` support has been extended to all supported GPUs.
- `optixGetTriangleVertexData` support has been fixed for large meshes.
- Fixed a crash on non-RTX GPUs when rendering with a refit IAS.
- New SDK samples.
    - optixCallablePrograms - Demonstration of *callable program* usage and SBT setup.
    - optixCurves - Minimal curve API usage example.
    - optixDenoiser - Demonstration of OptiX AI Denoiser usage.
    - optixDynamicGeometry - Shows typical setup for changing geometry between frames.
    - optixHair - More complicated curve API example.
    - optixNVLink - Demonstrates NVLink usage within an OptiX application.

What's New in 7.0

- 7.0 introduced the NVIDIA OptiX 7 API, a new low-level CUDA-centric API giving application developers direct control of memory, compilation, and launches while maintaining the programming model and shader types from previous versions of OptiX.
- Minimal host state is maintained. Scene graphs, materials, etc., are managed by the application rather than by OptiX.

- GPU memory is managed by the application using CUDA. (No OptiX buffers or variables)
- GPU launches are explicit and asynchronous using CUDA streams.
- Shader compilation is explicit. (Similar to DXR or Vulkan)
- All host functions are thread-safe.
- Source code for demand loading library is included and designed for direct inclusion in production applications.
- Multi-GPU operation is managed by the application.