NVIDIA® OptiX™ Ray Tracing SDK

Release Notes

Version 7.5.0 June 2022

Welcome to the 7.5.0 release of the NVIDIA OptiX SDK. This release adds sphere primitives, support for upscaling while denoising, and a preview of device code debugging features.

Upgrading to 7.5.0 may require source code changes, though they should be minimal. Applications compiled with the 7.5.0 SDK headers will require driver version 515 or later. Applications compiled with earlier SDK headers will continue to work, but recompiling with 7.5.0 will expose new features and may also improve performance.

System Requirements (for running binaries referencing NVIDIA OptiX)

Graphics Hardware:

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

Graphics Driver:

- NVIDIA OptiX 7.5.0 requires that you install an R515+ 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 run NVIDIA OptiX-based applications.

Development Environment Requirements (for compiling with NVIDIA OptiX)

- CUDA Toolkit 11.7

  This release has been tested with PTX generated from CUDA Toolkit 11.7. Other toolkit versions should also work, but 11.7 is recommended.

  Version 11.7 of the CUDA toolkit introduces the OptiX IR target with `--optix-ir` (see debugging, below).

  Version 11.1 of the CUDA toolkit introduces CUDA sparse textures, which are used in the NVIDIA OptiX Demand Loading library. This library is built only when the NVIDIA OptiX SDK is compiled with CUDA 11.1 or later.

- 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

- OptiX IR input for OptiX module creation
  - CUDA version 11.7 and above support generation of a proprietary binary format that conveys more information from NVCC to OptiX, including information required for debugging.
  - Invoke NVCC with `--optix-ir` to target this new format. Replacing `--ptx` with `--optix-ir` should be sufficient for most applications.
  - `optixModuleCreateFromPTX` will take this as input, similar to PTX input.
  - Inline PTX is still supported when using OptiX IR.
- Device code debugging preview
  - NSight Visual Studio Edition can show the value of variables inside of OptiX programs. In this preview version, some variables may not be present or have a value.
  - Only supported with OptiX IR input as described above.
  - Be sure to use `-G -O0` when compiling with NVCC to get debugging symbol information and use `OPTIX_COMPILE_DEBUG_LEVEL_FULL` and `OPTIX_COMPILE_OPTIMIZATION_LEVEL_0` in `OptixCompileOptions`.
  - A backtrace with only function name is still available with PTX input, similar to previous versions.
- New built-in sphere primitive
  - A geometry acceleration structure can now contain lists of spheres. Each sphere is specified by its center and radius.
  - On the host side, use the new build input type `OptixBuildInputSphereArray`. Similarly to curves, sphere intersection needs to be enabled by setting `OPTIX_PRIMITIVE_TYPE_FLAGS_SPHERE` in `OptixPipelineCompileOptions::usesPrimitiveTypeFlags`.
  - On the device side, sphere intersection returns one attribute. When the ray hits the sphere twice, the hit returned is the front face hit (entering the sphere), and the attribute contains the t value of the back face hit (exiting the sphere). Otherwise the attribute is 0. Note `optixIsFrontFaceHit` can be used to determine whether the hit is front face or back face.
  - The new device function `optixGetSphereData` can fetch any sphere's center and radius. Similarly to other get-data functions, it requires `OPTIX_BUILD_FLAG_ALLOW_RANDOM_VERTEX_ACCESS` to be set in `OptixAccelBuildOptions::buildFlags` before the acceleration structure is built. Setting this flag uses more memory.
  - See the SDK sample `optixSphere`.
- New error code `OPTIX_ERROR_DEVICE_OUT_OF_MEMORY`
  - Can be returned from previous asynchronous errors, or from `optixLaunch` when growing the stack memory required to support the launch.
  - In previous driver versions, an `OPTIX_ERROR_UNKNOWN` would be returned under these circumstances.
  - Memory allocation for stacks can be substantial when using large stacks; see `optixPipelineSetStackSize` to control this allocation.
- New OptixGeometryFlags enum
  - `OPTIX_GEOMETRY_FLAG_DISABLE_TRIANGLE_FACE_CULLING`
- Denoiser
  - A new mode combines 2X upscaling with denoising in a single operation. Use `OPTIX_DENOISER_MODEL_KIND_UPSCALE2X` or `OPTIX_DENOISER_MODEL_KIND_TEMPORAL_UPSCALE2X`.

- Denoiser scratch memory requirements are now lower in some cases. The required scratch memory sizes for the API functions `optixDenoiserComputeIntensity` and `optixDenoiserComputeAverageColor` should now be queried with `optixDenoiserComputeMemoryResources` instead of relying on the equations that were previously provided in the function header documentation. For `optixDenoiserComputeIntensity` and `optixDenoiserComputeAverageColor`, the scratch memory requirements are significantly lower. The scratch memory size for these functions is stored in `OptixDenoiserSizes::computeAverageColorSizeInBytes` and `OptixDenoiserSizes::internalGuideLayerPixelSizeInBytes`.
- Temporal denoising now relies on an internal guide layer to carry information from the previous denoising pass, specified with `previousOutputInternalGuideLayer` and `outputInternalGuideLayer` in the `OptixDenoiserGuideLayer` struct. A set of guide layers must be provided to the denoising pass as both input and output. These layers have the format `OPTIX_PIXEL_FORMAT_INTERNAL_GUIDE_LAYER` and should be sized according to the field `internalGuideLayerPixelSizeInBytes` returned by `optixDenoiserComputeMemoryResources`. See the programming guide for more information.
- `OptixDenoiserGuideLayer::previousOutputInternalGuideLayer` and `OptixDenoiserGuideLayer::outputInternalGuideLayer` must refer to two separate buffers, it is not possible to share memory between these buffers. After denoising a frame, these two buffers must be exchanged, so that `outputInternalGuideLayer` becomes `prevousOutputInternalGuideLayer` for the next frame. Instead of copying data into the previous guide layer, it is recommended to use a double-buffering strategy by swapping the content of the two `OptixImage2D` structs. Also use double-buffering for `OptixDenoiserLayer::output` and `OptixDenoiserLayer::previousOutput`.
- `OptixDenoiserParams::temporalModeUsePreviousLayers`
  - In temporal mode setting to 1 indicates the denoiser should read the values of the previous frame. Set to 0 for initial frames or when you want to reset the temporal sequence. In the first frame, when using temporal denoising modes, `OptixDenoiserGuideLayer::flow` must either contain valid motion vectors if available, otherwise the xy vectors must be set to zero (no motion). In temporal upscaling mode `OptixDenoiserLayer::previousOutput` is not accessed when `OptixDenoiserParams::temporalModeUsePreviousLayers` is not set.
- `OptixDenoiserSizes` struct has new fields
  - `computeAverageColorSizeInBytes`
  - `computeIntensitySizeInBytes`
  - `internalGuideLayerPixelSizeInBytes`
- `OptixDenoiserParams::denoiseAlpha` changed from a boolean flag to an enumerated type called `OptixDenoiserAlphaMode` which now has three possible values. These can be used for all denoiser models.
  - `OPTIX_DENOISER_ALPHA_MODE_COPY`: This is the default, alpha is copied from input to output and not denoised.
  - `OPTIX_DENOISER_ALPHA_MODE_ALPHA_AS_AOV`: This will apply the denoiser weights to the alpha channel treating it similarly to other AOV channels. This is the most efficient method to get a denoised alpha channel.
  - `OPTIX_DENOISER_ALPHA_MODE_FULL_DENOISE_PASS`: A new denoising mode that is useful where the alpha channel has a substantially different noise profile than the RGB image. If artifacts appear, such as extra halo areas around alpha, this mode may yield better denoised alpha channels. However, this will execute a separate inference pass on alpha, meaning that denoise execution will take twice as long.

- When debug exceptions are enabled, invalid ray exceptions are thrown for rays with zero length direction (with denorm values are treated as zero) and when tmin is negative.
- The many overloads of `optixTrace` have been replaced with a single templated function that can accommodate any number of payload values.
- Corrected documentation on the number of SBT records supported by a single GAS. The correct limit is 2^24. Improved the error message when this limit is exceeded.
- Compile time improvements
    - Improved concurrency when creating modules from PTX input in parallel. Improves compile times, and is especially noticeable when compiling many small modules.
- Fixed a bug where `optixAccelComputeMemoryUsage` and `optixAccelBuild` would change the current CUDA context.
    - Applications compiled against older versions of the NVIDIA OptiX SDK will see the context remain the same after the calls.
    - Applications compiled against 7.5 are required to have the current CUDA context be the context associated with the `OptixDeviceContext` when calling `optixAccelBuild` or any other OptiX function that invokes work on the GPU. For performance, the application should minimize the number of CUDA context changes around OptiX calls.
    - Validation mode will now catch when the CUDA context is incorrect. Undefined behavior or errors may result otherwise.
- Fixed a bug that sometimes led to a multiply defined symbol error when linking a pipeline with multiple built-in IS modules.

What's new in 7.4.0

- New compiler backend
    - Supports parallel compilation and other new features
    - This change should be invisible to applications, but if you have performance regressions, unexpected bugs or error messages that look like "New backend is missing implementation for PTX intrinsic <XYZ>", please contact us.
- Parallel compilation within a module exposed through new API
    - Exposes parallelism within a single module and between modules
    - Flexible threading model that can be used within existing application work queues
    - New API functions
        - `optixModuleCreateFromPTXWithTasks`
        - `optixModuleGetCompilationState`
        - `optixTaskExecute`
    - The following functions can now return `OPTIX_ERROR_ILLEGAL_DURING_TASK_EXECUTE`
        - `optixTaskExecute` if the task is already being executed on another thread
        - `optixModuleDestroy` if a task associated with the module is currently being executed
    - Added sample optixCompileWithTasks and SDK header CompileWithTasks.h that demonstrate this feature
- Payload types
    - To reduce register consumption in complex pipelines, OptiX 7.4 has introduced a mechanism to annotate the usage of each payload register. OptiX will analyze the lifetimes and reuse registers where possible.
    - The maximum number of payload registers has been increased from 8 to 32

    - These two features allow more workloads to improve performance by passing payload values in registers instead of local memory
    - Instead of supplying the number of payload registers in `OptixPipelineCompileOptions`, payload utilization information is now passed through `OptixModuleCompileOptions`. This contains an array of `OptixPayloadType` objects that annotate each payload value with their READ or WRITE usage.
        - The original mechanism of passing the number of payload registers in `OptixPipelineCompileOptions` will continue to work, though all modules in the pipeline must use the same mechanism
    - Refer to the programming guide for additional information
- Catmull-Rom round curve primitive type added
    - Same data layout as round quadratic b-splines
    - Access with new primitive enum values
        - `OptixPrimitiveType::OPTIX_PRIMITIVE_TYPE_ROUND_CATMULLROM`
        - `OptixPrimitiveTypeFlags::OPTIX_PRIMITIVE_TYPE_FLAGS_ROUND_CATMULLROM`
    - Control points can be retrieved on the device using `optixGetCatmullRomVertexData`
- Selectable endcap behavior for quadratic and cubic curves added, including the new Catmull-Rom curve
    - `OptixCurveEndcapFlags` used in `OptixBuildInputCurveArray::endcapFlags` and `OptixBuiltinISOptions::curveEndcapFlags`
    - Default is endcaps OFF. Previously the endcap behavior was ON.
- `OptixBuiltinISOptions` now has a `buildFlags` parameter that should match the ones supplied to `OptixAccelBuildOptions` when building the AS containing the built-in primitives
- When building acceleration structures containing curves with default build flags, an average of 7% reduction in memory was observed across a suite of datasets
- Demand Loading
    - An eviction mechanism now allows texture tiles to be reused when device memory is limited, allowing larger scenes to be rendered
    - Texture lookup can be skipped when texture gradients are large, using an optional base color that can be specified when the texture is initialized (either from metadata or from the coarsest miplevel)
    - Small textures (32x32 or less) no longer employ CUDA sparse textures, which saves memory and improves performance
    - UDIM UV mapping is now supported. A texture atlas can be constructed from a collection of sparse textures, each of which can leverage the small texture and base color optimizations.
    - API changes:
        - The EXRReader class has moved to a separate ImageReader library, with its own namespace
        - `createDemandLoader` no longer requires a vector of active device ordinals
- Denoiser
    - Added OptixDenoiserModelKind::OPTIX_DENOISER_MODEL_KIND_TEMPORAL_AOV
    - When tiling the denoiser the overlap is returned in the API, but you may notice the value has increased from 64 to 128
- OptixCompileDebugLevel

    - Removed `OPTIX_COMPILE_DEBUG_LEVEL_LINEINFO`
    - Added
        - `OPTIX_COMPILE_DEBUG_LEVEL_MINIMAL` - generate information that does not impact performance (currently generates line information useful for profiling)
        - `OPTIX_COMPILE_DEBUG_LEVEL_MODERATE` - generate some debug information with slight performance costs (currently generates a debug frame necessary to show a backtrace)
- Removed `OptixInstanceFlags::OPTIX_INSTANCE_FLAG_DISABLE_TRANSFORM`. This flag did not operate as intended, and users were already required to provide an identity instance transform.
- New Samples
    - optixOpticalFlow: demonstrates the use of the NVIDIA Optical Flow SDK to provide motion vectors for temporal denoising in lieu of application-provided motion vectors.
    - optixCompileWithTasks: demonstrates the use of the new parallel compile interface and contains a simple interface that can be adapted to many applications.
- Improved compile caching behavior with many concurrent accesses. If the database was locked, it would previously disable the cache.
- Name of the compilation database file has been changed to "optix7cache.db". The directory location has not changed.
- New warning issued when creating an `OptixDeviceContext` when the device memory size exceeds the host memory size
- PTX input that calls `__assertfail` no longer causes linking errors
- Launch parameter specialization now allows for single byte specialization regardless of whether the byte is part of a larger element such as int or float. This should reduce false errors when using specialization.
- Note: For OptiX 6 applications
    - Setting `RT_GLOBAL_ATTRIBUTE_ENABLE_RTX` to `0` via `rtGlobalSetAttribute` will result in an error as RTX is now the only supported mode for OptiX 6. The legacy megakernel execution strategy is no longer supported.
    - Name of the compilation database file has been changed to "optixcache.db", but the directory location has not.

Known Issues

1. Not all PTX instructions may be supported.
2. Enhanced line information for inlined functions available in CUDA 11.2+ may not always produce accurate results.
3. Pixel formats `OPTIX_PIXEL_FORMAT_UCHAR3` and `OPTIX_PIXEL_FORMAT_UCHAR4` are not supported by the Denoiser.
4. Concurrent launches from the same pipeline will serialize automatically on the device.
5. `OPTIX_COMPILE_DEBUG_LEVEL_FULL` does not currently generate debug information necessary for cuda-gdb or Nsight Compute VSE.

6. The demand loading samples trigger a run-time compilation error ('`identifier "__hisnan" is undefined`') when `CUDA_NVRTC_ENABLED` is set. This can be solved by setting `CUDA_MIN_SM_TARGET` to `sm_60` in when configuring with CMake.

What's new in 7.3.0

- Denoiser for temporal images and API simplifications
    - Temporal denoising is now supported with a new built-in model kind. In this mode a sequence of images can be denoised. The AI network was trained to reduce flickering for camera or geometry animations. It requires the denoised beauty image from the previous frame as input as well as flow (motion) vectors. Currently temporal denoising is not supported for AOVs.
        - Enable by using the new `OPTIX_DENOISER_MODEL_KIND_TEMPORAL` enum value from `OptixDenoiserModelKind`.
    - `optixDenoiserCreate` and `optixDenoiserSetModel` have been merged. `optixDenoiserCreate` now uses an enum to select the built-in model kind and `optixDenoiserCreateWithUserModel` that takes a user model.
    - `optixDenoiserInvoke`'s input has changed to specify guide layers specified explicitly in `OptixDenoiserGuideLayer` (e.g. albedo, normal or pixel flow layers) and separately from beauty layers which are now specified in `OptixDenoiserLayer`. `optixUtilDenoiserInvokeTiled` has similarly been updated to the new interface.
    - `OptixDenoiserOptions` changed to specify whether albedo and normal guide layers will be provided during `optixDenoiserInvoke`
    - New `OptixPixelFormat`
        - `OPTIX_PIXEL_FORMAT_HALF2`
        - `OPTIX_PIXEL_FORMAT_FLOAT2`
- Demand Loading
    - The `launchPrepare` and `processRequests` methods are now asynchronous, taking a CUDA stream argument on which operations are enqueued.
    - Requests for sparse texture tiles are now processed in the background by multiple CPU threads. This greatly improves concurrency, keeping the CPU busy with I/O and texture decompression while OptiX kernels perform rendering work on GPU.
    - Multiple streams are now supported. A degree of latency hiding can be accomplished by rendering a framebuffer as multiple tiles in round-robin fashion: while the texture data for one framebuffer tile is loading, work can proceed on subsequent tiles, returning to the first when its data is ready.
    - NVIDIA OptiX now provides the following texture footprint functions, which are used by the NVIDIA OptiX Demand Loading library to quickly determine which sparse texture tiles are required. These functions are hardware-accelerated on Turing and Ampere GPUs, with software emulation on older architectures.
        - `optixTexFootprint2D`
        - `optixTexFootprint2DLod`
        - `optixTexFootprint2DGrad`
- Improved curve intersectors

    - There is a new faster intersector for cubic and quadratic curves (based on Reshetov's Phantom intersector).
    - There is a new intersector for piecewise linear curves that is slightly faster and higher precision.
    - The new curve intersectors cull backfaces, treating the swept curve primitives as hollow. Rays originating inside the primitive will now exit the primitive.
    - Caveat: Internal rays can still hit internal end caps (invisible end caps between segments of a strand).
    - A bug in compaction of acceleration structures with curves has been fixed.
- New SDK samples:
    - optixDemandLoadSimple - Demonstrates a simple use of the Demand Loading library.
    - optixModuleCreateAbort - Compiles modules in separate processes, which can be interrupted.
    - optixMotionGeometry - Demonstrates motion blur for vertex positions, SRT transforms, matrix transforms, and combined motion.
    - optixVolumeViewer - Demonstrates incorporating OpenVDB volumes in an OptiX render, by using the open source NanoVDB library.
- Improved SDK samples:
    - optixDenoiser
        - New mode for temporal denoising
        - Tiling operation now supported.
    - optixDemandTexture - Now demonstrates the use of multiple streams for demand loading.
- `OptixPipelineCompileOptions` struct has changed. Please make sure to zero initialize this struct to avoid compilation errors.
    - `OptixPipelineCompileOptions pipelineCompileOptions = {};`
- Validation mode now checks the stream state before executing API functions that take a stream.
- New device function `optixGetInstanceTraversableFromIAS`
    - Return the traversable handle of a given instance in an Instance Acceleration Structure (IAS). This handle is not directly traversable, but can be used to query instance transforms and other information.
- New OptixBuildFlag `OPTIX_BUILD_FLAG_ALLOW_RANDOM_INSTANCE_ACCESS`. This flag is required to call `optixGetInstanceTraversableFromIAS`.
- New device function `optixGetInstanceChildFromHandle`
    - Returns child traversable handle from an `OptixInstance` traversable
- New exception codes that can be triggered by `optixGetTriangleVertexData` and `optixGetInstanceTraversableFromIAS` when exceptions are enabled.
    - `OPTIX_EXCEPTION_CODE_INVALID_VALUE_ARGUMENT_0`
        - The value passed in `ias` to `optixGetInstanceTraversableFromIAS` is not a valid Instance AS.
        - The value passed in `gas` to `optixGetTriangleVertexData` is not a valid Geometry AS.
    - `OPTIX_EXCEPTION_CODE_INVALID_VALUE_ARGUMENT_1`
        - The `index` passed is out of range.
    - `OPTIX_EXCEPTION_CODE_INVALID_VALUE_ARGUMENT_2`
        - The `sbtGASIndex` passed to `optixGetTriangleVertexData` is out of range.

    - OPTIX_EXCEPTION_CODE_UNSUPPORTED_DATA_ACCESS
        - optixGetTriangleVertexData was called on an acceleration structure that was built without OPTIX_BUILD_FLAG_ALLOW_RANDOM_VERTEX_ACCESS set.
        - optixGetInstanceTraversableFromIAS was called on an acceleration structure built without OPTIX_BUILD_FLAG_ALLOW_RANDOM_INSTANCE_ACCESS set.
        - An acceleration structure built with motion was used in a pipeline without motion enabled.
- If the OptiX compile cache is corrupted, OptiX will now attempt to delete and reinitialize the cache.
- Added OPTIX_CACHE_MAXSIZE environment variable to control the size of the disk cache.
- AS builds have been optimized for faster trace times.
- PTX must be recompiled with the new SDK headers if the host uses the new SDK.
- NVRTC is no longer enabled by default when compiling the SDK samples. It can be enabled by setting the CMake variable CUDA_NVRTC_ENABLED.
- Bug fixes:
    - Fixed bug to allow optix_stack_size.h to be able to be included in more than one compilation unit.
    - Fixed bug with surf2Dwrite calls having no effect.
    - Fixed bug with some denormal floats being treated as zero.
    - Fixed some bugs that would prevent correct line information from being used in profiling and debugging.

What's New in 7.2.0

- 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 NVIDIA 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

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 NVIDIA 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 NVIDIA 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 NVIDIA 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.