NVIDIA(R) APEX(TM) SDK Release Notes

Note: APEX SDK has been deprecated

Contents:

APEX 1.4.1 - Release Notes (APEX has

Note: APEX SDK has been deprecated

Installing APEX

Windows Requirements:

1. Visual Studio 2010, 2012, 2013 or 2015.

2. Microsoft DirectX SDK June 2010 or later (to build the Windows APEX samples)

    1. The DirectX include path should be set in Visual Studio in VC++ Directories to build the Windows samples
    2. The DirectX End-User Runtime Web Installer (dxwebsetup.exe) can be downloaded from Microsoft if the prebuilt APEX samples do not run because they cannot find the June 2010 DX runtimes.

3. Graphics renderer must support at least OpenGL version 1.5 to run the Windows APEX samples in OpenGL mode

APEX for PhysX 3.4 Requirements

1. PhysX SDK for 3.4 for Windows (the APEX source distribution requires the PhysX source distribution)

SDK Zip Distributions:

The source and binary distributions come in compressed zip distributions. The source distribution does not contain samples, just the core APEX SDK.

> **Note:** To build the APEX projects, the PhysX SDK path in the nxpath.vsprops files must be modified. The UpdatePhysXPaths.vbs script is provided for this purpose. It will modify the nxpath.vsprops files under the compiler folders. Similar Android distribution setup is done by setting

up Makefile without using this script. The 3.x PhysX path should contain the folder location (absolute or relative) that contains the “Include” directory.

Note: The APEX SDK source build requires the PhysX 3.4 source distribution.

New in 1.4

General

- Refactored public API naming convensions
- Removed global LOD support
- Refactored APEX samples and added APEX snippets
- Removed PhysX 2.8.4 support. Added build configuration without PhysX linking for Clothing module

Optimizations

Fixes

Known Issues

New in 1.3.3

General

- The APEX release build configurations for x86 and x64 now also contain authoring code. On other platforms authoring is still removed for release builds.
- Windows APEX samples now render using D3D11 by default. Use the -d3d9 command line option to render using D3D9 instead.

Optimizations

Fixes

Known Issues

- PVD support may not be working on Android.

New in 1.3.2

General

- Added NVTX profiling support for non-release builds. Traces can be captured using Parallel Nsight Analysis Tools. NVTX is supported only for Windows and Android.

Optimizations

- Added ability to change render resource lock behavior or disable locking altogether, to improve performance. The setting is per module. See NxModule::setRenderLockMode and NxApexRenderLockMode.
- Added ability to disable potentially expensive stat collection if it’s not needed. See NxApexSDK::setEnableApexStats.

Fixes

- Fixed crash if application creates APEX SDK after already connecting to PVD.

Known Issues

- PVD support may not be working on Android.

New in 1.3.1

General

- Added VS2012 support for source distributions with PhysX 3.x.
- Assertions are now enabled in the checked configuration.
- Added NxApexActor::setEnableDebugVisualization to selectively disable debug visualization on particular actors.
- The NxUserRenderResourceManager's surface buffer create and release virtual methods must be implemented by the user. They are only used by particular turbulence features, so if turbulence isn't being used the implementation can be empty.
- Starting with NVIDIA R302 drivers, application developers can direct the Optimus driver at runtime to use the High Performance Graphics to render any application - even those applications for which there is no existing application profile. The APEX samples now makes use of this "NvOptimusEnablement" feature to enable High Performance Graphics by default.
- Debug info added to all PS4 builds except release builds.
- Android sample builds now generate an APK. This requires some extra defines for Java and Ant. See the Android examples section of the sample documentation for details.

Fixes

- Fixed rare Named Resource Provider failure to find resource.
- Using allocViewMatrix(ViewMatrixType::USER_CUSTOMIZED) now works.
- Cleaned up several PS4 compiler warnings.

Known Issues

- PVD support may not be working on Android.

New in 1.3

General

- The legacy modules are all combined in one module, APEX_Legacy.
- The particle modules are all combined in one module, APEX_Particles (with the exception of Turbulence).
- The Wind module was removed.
- The NxUserRenderBoneBufferDesc::maxBones member will now reflect the actual max bones specified in NxUserRenderResourceManager::getMaxBonesForMaterial(). The actual number of bones used will be sent in the NxUserRenderBoneBuffer::writeBuffer() numBones parameter. In previous APEX versions, NxUserRenderBoneBufferDesc::maxBones reflected the minimum(numBones, maxBones).
- APEX's output stream will automatically be sent to PVD when the application is connected to PVD.
- The outputStream member of the NxApexSDKDesc has been removed when using PhysX 3.x. APEX will automatically use the same output stream as PhysX.

Known Issues

- PVD support may not be working on Android.

New in 1.2.4

General

- EditorWidgets header files added to binary distributions.
- Added more asset previews for consistency.
- PS3 gcc support has been deprecated in APEX 1.2.4 in favor of SNC. The gcc version is still included but not supported.
- Fixed some allocations that were not going through the user-supplied allocator.

Known Issues

- There are shadow and flickering issues when running the APEX samples using the DirectX 11 renderer (–d3d11).
- Some key and button mappings in the APEX samples are incomplete on PS3 and Xbox360.
- There may be issues connecting to PVD from consoles with the PhysX 2.8.4 build of APEX.

New in 1.2.3

General

- NxResourceProvider is now case-insensitive by default. To get the old case-sensitive behavior, set resourceProviderIsCaseSensitive true in the NxApexSDKDesc.
- Fixed issues with re-entrant read locks described below.
- GPU rigid bodies are no longer supported on pre-Fermi hardware.

Known Issues

- There are shadow and flickering issues when running the APEX samples using the DirectX 11 renderer (–d3d11).
- Some key and button mappings in the APEX samples are incomplete on PS3 and Xbox360.
- There may be issues connecting to PVD from consoles with the PhysX 2.8.4 build of APEX.

New in 1.2.2

General

- APEX requires the use of the PhysX 3.x scene multiple-reader-single-writer lock to access both APEX and PhysX scene objects when using the *PxSceneFlag::eREQUIRE_RW_LOCK* flag. This allows the application using APEX to access PhysX objects from multiple threads. For more information, see the “Scene Locking” section of the “Data Access and Buffering” PhysX 3.x documentation. (See Known Issues for limitations when using this lock)

Known Issues

- There are shadow and flickering issues when running the APEX samples using the DirectX 11 renderer (–d3d11).
- As of PhysX 3.2.2 the lock offered by the PhysX scene does not support re-entrant read locks. This is currently a problem if the application attempts to use the APEX actor or asset APIs while APEX and PhysX are simulating (for instance, raycasting against an APEX Destructible actor using NxDestructibleActor::raycast). This will be addressed in a future version of APEX.
- Some key and button mappings in the APEX samples are incomplete on PS3 and Xbox360.
- There may be issues connecting to PVD from consoles with the PhysX 2.8.4 build of APEX.

New in 1.2.1

General

- NxApexSphereShape getRadius/setRadius use PxF32 instead of PxVec3 to represent the radius.

Known Issues

- Some key and button mappings in the APEX samples are incomplete on PS3 and Xbox360.
- There may be issues connecting to PVD from consoles with the PhysX 2.8.4 build of APEX.

New in 1.2

Visual Studio 2010 Support

- Both the APEX SDK source and sample projects are distributed solely as Visual Studio 2010 solutions

NxParameterized

- Added AndroidARM target.
- Better error checking in xml serializer.
- Unified behaviour of initParamRef() and setParamRef().

Render Mesh Asset

- Tangent support the float4 format now (making bitangents unnecessary)

APEX Force Field

- Existing force field module renamed to field boundary.
- New force field module which works with APEX Particles and PhysX 3.x objects.

General

- Added an optional completionTask to NxApexScene::simulate that will be called when fetchResults is ready to be called.
- The ‘pxtask_cuda’ DLL, which contains the CUDA Context Manager and associated CUDA PxTask functionality, was renamed ‘PhysX3Gpu’.
- APEX no longer supports using apexuser.dll to override the file stream returned from NxApexSDK::createStream.

- APEX stats are reported in milliseconds instead of seconds. This was a change done very late in the 1.1 product cycle, so when upgrading from an early 1.1 this is still new in 1.2.
- NxApexScene::acquirePhysXLock() is now available so that both APEX and the application may access the PhysX 3.2 scene during simulation without performing concurrent write operations. PhysX 3.2 allows “multiple reader/single writer” access, but this has not yet been implemented within APEX.
- NxApexScene::simulate() has a new parameter called ‘scratchMemBlock’, which may be used by PhysX 3.2 for temporary data during simulation.
- The Wind and Explosion modules are deprecated and may be removed from a future APEX release. They do not support PhysX 3.2.
- PS3 gcc support is deprecated in APEX PS3 source distributions. All PS3 gcc support may be removed from a future APEX release in favor of SNC.

Known Issues

- Some key and button mappings in the APEX samples are incomplete on PS3 and Xbox360.
- There may be issues connecting to PVD from consoles with the PhysX 2.8.4 build of APEX.

New in 1.1

NxParameterized

- New method Interface::clone.
- Handles which are constructed from const Interfaces do not allow to change those Interfaces via setParamXxx.
- Generic reference visitor NxParameterized::getReferences
- Streamed non-inplace binary deserialization
- Fixed handling of defaultValue in structs
- Fixed detection of hint types
- Automated versioning of legacy classes
- Customizable order of legacy objects upgrade instead of fixed bottom-up
- Custom alignments of struct fields
- ParamTool can now print summary of file contents

Serialization

- ParamTool does not support legacy asset formats anymore
- Changed some error codes

Render Mesh Asset

- setOverrideMaterial was added to allow switching of material directly at runtime.
- Debug rendering vor vertex normals and tangents.

Use of Cuda 4.2

- Necessary for supporting latest hardware.

New in 1.0

Use of Cuda 3.0

- This allows using interop

Use of a new Foundation (Px prefix)

- No more 3x4 matrix, replaced by a true 4x4 column wise matrix.

Serialization

- Apex 0.9 binary serialization (.aca, .pda, .arm files) are deprecated. Code to serialize assets to these old formats has been removed. Code to read these old formats is still in place but also deprecated. It will likely be removed for 1.1.
- As a replacement we are using NxParameterized objects that can be serialized to either xml (.apx) or binary (.apb) files, where binary files can be optimized for specific platforms.
- Renamed NxParameterized::NxSerialiser to NxParameterized::Serializer

Render mesh asset

- The authoring pipeline was revised.
- Each vertex buffer semantic can now use a wide range of possible formats.
- Added compressed render data formats like BYTE_UNORM4 etc.
- Changes to instance buffer semantics and formats: removed old POSE format which used Mat34, and now use ROTATION_SCALE (Mat33) and POSITION (Vec3) in its place. Removed unused instance buffer semantics.
- Opaque meshes can replace render mesh assets. An Opaque mesh is a

void pointer to a user implemented render mesh that APEX does not know or understand about. It will still issue render calls using the (opaque) void pointer.

NxModule

- Generic module generation: Get rid of the various descriptors for Modules and unifiy them as parameterized objects.

NxApexAsset

- added releaseAndReturnNxParameterizedInterface() which allows reuse of the parameterized object after object destruction.

NxApexRenderable

- removed lod parameter from dispatchRenderResource method. All modules that support

NxApexScene

- added finalstep to NxApexScene::simulate() indicating that manual substepping is used and this is the last substep.

Debug Rendering

- Removed all debug rendering from NxApexSDK object.
- Added parameterized objects to handle debug rendering configuration for the framework and each module separately.
- Changed code to set up the view and projection matrix.
- Parameterized interface to customize debug rendering colors.

APEX Clothing Release Notes

APEX Clothing 1.3.3

New Features

Removed

Optimizations

Fixes

Known issues

- If asyncFetchResults is false, the application must have more than one worker thread to avoid the possiblity of deadlock.
- The RecomputeSubmeshes visualization flag is not functional.
- It is not deterministic whether the new or the old result is returned in acquireRenderProxy on NxClothingActor, if it is called between simulate and fetchResults, when running with PhysX 3.x.

APEX Clothing 1.3.2

New Features

- It is now safe to call updateRenderResources on the render proxy in parallel, for example in the render thread, if the APEX SDK parameter ‘renderMeshActorLoadMaterialsLazily’ is set to false.
- Added support of partial custom physical meshes.

Removed

- Physical LOD is deprecated. Will be removed next version.

Optimizations

- Reduce the amount of data sent to PVD when clothing is in a scene.

Fixes

- Fixed friction for GPU clothing.
- Fix a race condition when running several clothing scenes in parallel.
- Fixed clothing render proxy: now always return old result before fetch has been called.
- Fixed self-collision jittering on scaled down actors: scale restpositions. Fixed debug rendering when scaled.
- Fixed crash in cloth cooker.
- Fixed initialization pose of clothing collisions when clothing actor uses local space simulation.
- Fixed crash when a clothing actor with a NxClothingTriangleMesh switches LoD.
- Fixed scaling of cloth collision planes.

Known issues

- If asyncFetchResults is false, the application must have more than one worker thread to avoid the possiblity of deadlock.
- The RecomputeSubmeshes visualization flag is not functional.
- It is not deterministic whether the new or the old result is returned in acquireRenderProxy on NxClothingActor, if it is called between simulate and fetchResults, when running with PhysX 3.x.

APEX Clothing 1.3.1

New Features

- Added NxClothingActor::getSimulationVelocities()
- ClothingTool: Some names were changed to match DCC plugin naming conventions.
- Clothing triangle collision debug rendering: render ignored triangles red.
- Clamp to 500 collision triangles on GPU to avoid hitting shared memory limit.
- Added API to add and remove triangles from an NxClothingTriangleMesh at runtime.
- Added friction to convex collisions against cloth.
- No longer deprecated: updateMaxDistanceScale, getPose, setWind, setMaxDistanceBlendTime, getMaxDistanceBlendTime, setLODWeights

Removed

- Removed unused API NxClothingUserRecompute.
- Unused zeroStretchStiffness parameter has been removed.

Fixes

- Fixed incorrect scaling in NxClothingAssetAuthoring::applyTransformation.
- Fixed infinities generated in NxClothingAssetAuthoring::applyTransformation.
- Fixed bug that could cause clothing material to be set every frame.
- Fixed race condition when creating/deleting cloths in parallel.
- NxClothingActor methods getSimulationPositions and getSimulationNormals now block when asyncFetchResults is set to

true.

- Fixed crash when calling releaseGraphicalData on a clothing asset (applies if application does its own skinning)
- Fixed a 3x clothing cooker crash.
- Fixed a bug where APEX could wake frozen cloth, causing APEX to output a “Cloth has not stayed static” message.
- Fixed an issue when using multiple GPU dispatchers with GPU clothing.

Known issues

- If asyncFetchResults is false, the application must have more than one worker thread to avoid the possiblity of deadlock.
- The RecomputeSubmeshes and RecomputeVertices visualization flags are not functional.
- It is not deterministic whether the new or the old result is returned in acquireRenderProxy on NxClothingActor, if it is called between simulate and fetchResults, when running with PhysX 3.x.

APEX Clothing 1.3

New Features

- New render API: NxClothingRenderProxy to allow easier rendering with render thread (buffered render data)
- Added PVD object debugging support for clothing.
- Visualization of clothing in PVD is possible by connecting to PVD with object debugging enabled, and then enabling “Constraints” visualization in the PVD Preferences.

Fixes

- Clothing GPU memory usage now included with the rest of the heap stats from the CudaMemoryManager.
- Changed asyncFetchResults to true by default to improve performance in many cases and avoid deadlock.
- Fixed debug visualization of the PhysX 3x clothing material parameters. Added visualization of compressionRange and stretchRange parameters and length and cross-section fibers.

Known issues

- If asyncFetchResults is false, the application must have more than one worker thread to avoid the possiblity of deadlock.
- The RecomputeSubmeshes and RecomputeVertices visualization flags are not functional.

APEX Clothing 1.2.4

Fixes

- Fixed issues with teleport and reset.
- Fixed a crash in tangent computations.
- Fixed an issue where tangent results were different for skinned tangents.

Known Issues

- Summer dress sample (SimpleClothing scene 3) behaves differently on the GPU compared to the CPU.

APEX Clothing 1.2.3

New Features

- Added solver iteration count to VISUALIZE_CLOTHING_SOLVER_MODE debug rendering.

Fixes

- Fixed an issue with clothing simulation when simulate(0) is called.
- Fixed a bug with normals for the case where assets have a significant amount of mesh-mesh skinned vertices
- Fixed a simulation bug where collision failed for vertices close to the cloth bounds
- Fixed a race condition when a lot of actors of the same asset are switched on and off (manually or by lod)

Removed

- Removed support for the embedded 3x clothing solver on pre-Fermi GPUs.

APEX Clothing 1.2.2

New Features

- New ClothingMeshSkinningMap API to get mesh skinning information, so it can be done in the application.
- New API to add collision objects to a clothing actor. This can be used to implement collision with world objects.

APEX Clothing 1.2.1

New Features

- Support for convex collision added to 3x clothing solver.

  Limitation: only a total of 32 planes per clothing actor is supported.

- Self collision parameters moved to the ClothingMaterialLibrary NxParameterized interface. Removed NxClothingAssetAuthoring::setSimulationSelfcollision() and NxClothingAssetAuthoring::setSimulationSelfcollisionThickness().

Fixes

- Fixed a Clothing Tool issue where the “Generate Tangent Space” button could generate several warning dialogs for certain older clothing assets.

Known Issues

- Running multiple NxApexScenes simultaneously is only supported with the 3x solver. The 2.8 solver can generate writelock NxScene access failures.
- The new 3x cloth solver does not send debug visualization data to PhysX Visual Debugger.
- GPU memory allocated by the 3x clothing solver is not reported in the heap stats from the CudaMemoryManager.

APEX Clothing 1.2

New Features

- Support for PhysX 3x cloth solver. Even builds with PhysX 2.8.x will be able to run the new solver if the ClothingPhysX3 module is present (will be autoloaded).
- Removed bitangent semantic for rendering. All rendered data now uses float4 for the tangent semantic, tangent.w being the sign of the cross product of normal and tangent.
- Local space simulation. The user can either chose a local space or specify a bone in the hierarchy to which the simulation will be done locally.
- Tapered capsules as collision volumes. This will only work with PhysX 3x solver.
- Velocity shader can take positions and velocities now. Velocities can be modified by the shader.
- Added several PhysX 3x exclusive clothing material parameters: Compression & streching stiffness for stretching, bending and shearing constraints, inertia scale & drag (for localspace sim), mass scale for better collision handling.
- Turned ‘parallelize fetch results’ on by default (in module descriptor) and added functionality to run the additional fetch results work delayed until the first actor gets rendered.

Known Issues

- Running multiple NxApexScenes simultaneously is only supported with the 3x solver. The 2.8 solver can generate writelock NxScene access failures.
- The new 3x cloth solver does not send debug visualization data to PhysX Visual Debugger.
- The “Generate Tangent Space” button in Clothing Tool can generate

several warning dialogs for certain older clothing assets. (Fixed in 1.2.1)

APEX Clothing 1.1

New Features

- Per-Actor scaling. Each actor can have an individual scale and simulates properly.
- Asynchronous cooking. When several actors with different per-actor scale are created, cooking will be delayed if another actor already started cooking. While waiting, actors will not be simulated.
- Manual substepping: A PhysX scene that will run multiple substeps is now handled properly by clothing. Interpolated simulation meshes will be generated for each individual substep to remove simulation artifacts caused by substepping. Caution: This feature causes problems if used together with 2.8.x physX particles. When enabled, APEX calls simulate/fetchResults on the corresponding physX scene several times. PhysX particles have buffers that need to be read from the application after each fetchResults (deleted particle IDs), which is not possible in this case. Use allowApexWorkBetweenSubsteps to enable/disable the feature.
- Teleport without reset: Clothing Actors can now be teleported without all the vertices being reset to the animated positions. This will only lead to good results if only the pose of the actor and not the state of the animation is changed.
- Frozen state: Clothing can be frozen in a given state and will cease simulation. It can then be woken again in the same state. This contrasts the regular way where re-enabled clothing starts from the animated state instead.
- Velocity Callback: A user callback that allows to read and write the velocity values of every simulated vertex. Can be used to play sound depending on certain changes in velocity or to implement i.e. a wind effect.
- Support for morph targets (also known as blendshapes): At actor creation a set of vertex displacements can be used to modify the mesh.

- Platform Tags: Each graphical LOD can have a list of strings attached. When converting the .apx/.apb files to a given platform, LODs that don’t match a certain pattern can be removed based on this. This allows for reducing asset size by removing LODs that are unsuited for a particular platform.
- Correct Simulation Normals: Vertices that are on the border of the simulated and non-simulated part of the physical mesh can have wrongly calculated normals. This setting will try to correct for that.
- Adaptive Target Frequency: Reduces the high frequency jittering that happens due to slightly varying timesteps.
- Pressure: Closed cloth meshes can be filled with pressure.
- Vertex Velocity Clamp: Adds a maximum velocity in all 6 major axis and clamps all velocities to those.

Improvements

- Switching of graphical LoDs has been improved in respect to copying position and velocity from the old to the new simulation mesh.

Removed

- Legacy serialization is gone. Any old .aca file cannot be loaded anymore. They need to be converted to .apx/.apb using the 1.0 ParamTool instead.
- Some API marked as deprecated has been deleted such as NxClothingActor::setFlags().
- Removed separate NxClothingMaterialLibrary (.acml files) as an asset type. Materials are now integrated into the Clothing Asset file directly.
- Parallel physics mesh skinning and parallel mesh-mesh skinning. The frame delay that was introduced by this feature made it useless to most applications.

APEX Clothing 1.0 Beta

New Features

- New Mesh-Mesh skinning algorithm that is faster and produces more reliable results.
- Max Distance can be scaled per clothing actor on top of all the physical lod features.

Improvements

- Using per-actor Tasks to allow more fine grained parallelization.
- NxClothingActor visibility can be toggled to save unnecessary computations in case where the actor is not visible.

Removed

- Replaced NxClothingActorDesc and NxClothingPreviewDesc with parameterized objects.
- Clothing Material Library has been deprecated. The separate asset is gone and has been merged with the clothing asset. The classes are still in place to load the .acml files, which get merged into the clothing asset immerdiately after the .aca has been loaded. After that both files can be replaced by a single clothing asset .apx/.apb

Known Issues

- Clothing does not support running substeps. When doing so it will emit a warning and then just run as good as possible. This can lead to artifacts where vertex velocities can sometimes behave erratically.
- NxClothingAssetAuthoring::setExportScale is not working anymore as it was designed to work with the removed binary serialization. As a replacement, NxClothingAssetAuthoring::applyTransformation() can

be called before serialization, it apply a transformation and/or a scale to the asset, which can then be serialized with the new serialization.

APEX Particles Release Notes

APEX Particles 1.4

New Features

- Refactored render interface for Particles and TurbulenceFS

APEX Particles 1.3.3

New Features

Removed

Improvements

Fixes

- Fixed issue where PhysX monitor could keep dynamic rigid bodies awake.

Known Issues

- CUDA/D3D9 interop is unsupported.
- Added experimental velocity sources to turbulence simulation, but there are known issues. It’s not recommended to use this feature.

APEX Particles 1.3.2

New Features

- Weighted field sampler collision filtering was added (see migration guide).
- Added RateVsEmitterDuration curve to APEX emitters.
- Added minSamplingFPS parameter to APEX emitter assets to reduce discontinuities of fast moving emitters.
- Added a pose matrix semantic to NxUserRenderInstanceBufferDesc.
- Added topSphericalForce and bottomSphericalForce to VortexFS.
- Added scaling of APEX emitters and field samplers. See setCurrentScale/getCurrentScale methods for attractor, vortex, windFS, emitter, effect package, heat source, substance source, and turbulence actors.
- Added collision filtering for heat sources and substance sources. See fieldSamplerFilterDataName parameter.
- Added OrientScaleAlongScreenVelocity IOFX modifier.

Removed

- Removed ‘collisionFilterDataName’ from Turbulence, ParticleIOS, and BasicIOS, since it is redundant under PhysX 3.x. Now just use ‘fieldSamplerFilterDataName’
- Removed ‘fieldBoundaryFilterDataName’ from various field sampler assets since it is now considered deprecated.
- Removed obsolete semanticFormats[] from NxUserRenderInstanceBufferDesc.
- Removed support for GPU acceleration on pre-Fermi GPUs.

Improvements

- HeatSourceActor/SubstanceSourceActor NxParameterized parameter

for initial position was renamed to ‘initialPose’ and its type was changed to MAT34.
- HeatSourceActor/SubstanceSourceActor NxParameterized parameter for initial scale was renamed to ‘initialScale’.

Fixes

- Under PhysX 3.x all collision filtering related to both particles and field samplers now uses the standard PxFilterData and SimulationFilterShader callback.
- Resolved several issues where it was possible to exhaust GPU constant memory under certain heavy loads. APEX will now continue to work under these conditions, although performance may be slightly affected.
- Fixed crash that could occur when using turbulence debug visualization.
- Fixed particle time stats when using CPU particles.
- Fixed uninitialized render volume in EffectPackages. Could lead to particles not getting rendered.
- Fixed possible turbulence crash when using the updatePerFramesRange feature.
- When using opaque mesh rendering, userRenderData is now correctly passed to the application through the NxUserRenderResourceDesc.
- Fixed bad reference counting of opaque meshes.
- Fixed rare race condition that could cause some particles not to be rendered.
- Fixed synchronization issues with PhysX when using ParticleIOS.
- Fixed error in particle benefit calculation that could inject NANs into code.
- Fixed bad LOD calculation with updatePerFramesRange feature.
- Fixed race condition in turbulence.
- Fixed improper render mesh release when using IOFX mesh particle rendering.
- Fixed issues with the turbulence dissipation time feature.

Known Issues

- CUDA/D3D9 interop is unsupported.
- Added experimental velocity sources to turbulence simulation, but there are known issues. It’s not recommended to use this feature.

APEX Particles 1.3.1

New Features

- New WindFS field sampler for wind effects.
- Add ‘fieldDragCoeff’ & ‘fieldWeight’ parameters to all assets in BasicFS
- Turbulence dissipation time, allows to remove turbulence vortices in a predetermined time. But see known issues.
- Added ability to sample a TurbulenceFS’s velocity field directly to a surface buffer.
- The NxUserRenderSurfaceBuffer::writeBuffer parameters have changed to support writing to offsets in any of three dimensions.
- Object scale for emitter actors. (need details)
- More efficient use of CUDA constant memory in IOFX

Removed

- Removed useless fuctions (addFilterData, removeFilterData, getFilterData) from NxForceFieldActor. PxFilterData can be specified in NxForceFieldActorDesc.

Fixes

- Fixed crash when a turbulence asset had maxCollidingObjects set to 0.
- Fixed BasicIOS and TurbulenceFS not honoring all collision filtering with PhysX 3.x. Note: now that the collision filtering is correctly handled, the behavior could change if you were using non-default collision filtering.
- Fixed an issue with VortexFS not working due to an uninitialized variable.
- Fixed a bad interaction between a rotating capsule and a turbulence grid.

- Optimized multigrid/diffusion in Turbulence
- More robust handling of CUDA out of memory for TurbulenceFS actor.
- Fix a ParticleIOS crash when an emitter is deleted.
- Various EditorWidgets fixes.
- Fix for incorrect EffectPackage duration calculation.
- Fixed incorrect capsule orientation when colliding with BasicIOS and TurbulenceFS.
- APEX now correctly takes into account total elapsed time, which fixes time-dependent behavior of emitters.
- Fix for NULL pointer access in EffectPackageActor.
- Re-enabling a disabled effect in an EffectPackageActor using setEffectEnabled=true now workes.
- Fixed issues with turbulence debug visualization.

Known Issues

- CUDA/D3D9 interop is unsupported.
- When using the new turbulence dissipation time feature, some other field samplers may appear to start late. Also cleaning time may not work. These issues will be fixed in a future release of APEX.

APEX Particles 1.3

New Features

- Particle and field sampler modules including IOFX, Emitter, BasicIOS, ParticleIOS, and BasicFS have been merged into one “Particles” module. TurbulenceFS is still a separate module.
- APEX now supports Effect Packages, a collection of particle-related assets that can be instantiated together in an applications. APEX provides the Particle Effect Tool for authoring.
- APEX now requires Sprite and Instance Buffers implementations to specify their layout (NxRenderSpriteLayoutElement and NxRenderInstanceLayoutElement) by using semantic/format specification defined in corresponding enumerations.
- Added 2 counters to APEX IOFX Stats - SimulatedSpriteParticlesCount, SimulatedMeshParticlesCount
- Added Noise and Vortex Field Samplers in the BasicFS module (now in Particles module).
- APEX Emitter Actor now provides an optional user emitter position validation callback (NxApexEmitterActor::setApexEmitterValidateCallback) used to allow the application to prevent particles from being emitted in invalid locations (for instance, on the other side of a wall).
- APEX Sphere and Sphere Shell Emitter now supports ‘hemisphere’ property (allows using only specified sphere cap as emitter).
- APEX requires minimal GPU Compute Capability 1.1 now (G92 or better GPU).
- Changed ‘min’ ‘max’ hints for APEX Particle IOS Asset to ‘uimin’ and ‘uimax’ so they are treated as hints only for the user-interface code; but will not ‘invalidate’ assets.

Removed

- Removed fluidVelocityMultiplier and fluidVelocityClamp parameters from TurbulenceFS asset.
- Removed redundant parameter ‘position’ from APEX Heat Source

Fixes

- APEX Turbulence moving grid behaviour is fixed.
- CUDA/D3D11 interop is now working. Note that CUDA/D3D9 interop still has known issues and is not supported or recommended.

Known Issues

- CUDA/D3D9 interop is unsupported.
- Mixing interop and non-interop IOFX rendering at the same time is unsupported.

APEX Particles 1.2.4

Fixes

- Fixed a race condition issue due to improper scene locking with PhysX 3.2.x in turbulence.
- Fixed GPU memory leak with turbulence/convex collision.
- Fixed an LOD issue with ParticleIOS that could cause a crash.

Known Issues

- (PhysX 3.2.4 only) You may see the nuisance error stream warning “Adding particles before the first simulation step is not supported” when using ParticleIOS. This warning may be ignored.

APEX Particles 1.2.3

New Features

- IOFX sprite rendering was reworked to allow writing sprite rendering data more efficiently. See the new NxUserRenderResourceManager::getSpriteTextureData() and NxUserRenderSpriteBuffer::writeTexture() callbacks and the new NxUserRenderSpriteTextureDesc.
- Added NxApexEmitterActor::getSimParticlesCount() to get the number of simulated particles. Works with BasicIOS and ParticleIOS.
- Added NxApexEmitterActor::getRateRange().
- Added ColorByVelocity IOFX modifier.
- Added includeVerticalDirection rotation modifier to control whether vertical speed affects rotation.
- Added getPose() and setPose() methods for NxHeatSourceActor.

APEX Particles 1.2.2

New Features

- Added Turbulence Heat Source assets and actors instead of them being only part of the Turbulence module C++ API
- Added setEnabled API to JetFS and AttractorFS to enable/disable a field sampler actor at runtime
- BasicIOS: Added NxParameterized collisionWithConvexMesh and collisionWithTriangleMesh to BasicOS to enable/disable different shape collision
- Added fluidViscosity to TurbulenceFS.

Fixes

- Added 64bit collision filtering between the ParticleIOS (PhysX 3.x particles) and APEX Field Samplers (in the fieldSamplerFilterData member of ParticleIosAssetParam)
- Changed grid resolution type to an enum. Before it was a float, so it was possible to set invalid values that would cause errors.

Improvements

- Particle rendering is now done with a single render buffer for non-interop and a double render buffer for D3D-CUDA interop. Previously it was always double buffered and possibly triple buffered for interop. Note that the placement of the NxApexScene::prepareRenderResourceContexts() changes from just before calling the UpdateRenderResources methods to outside of the simulation (after FetchResults).
- EditorWidgets: Flipped “Life Remaining” axes to show “Life Time” instead, based on user feedback.

APEX Particles 1.2.1

New Features

- Added noise feature to turbulence grid simulation.
- Added drag coefficient parameter to turbulence grids.
- Attractor field sampler works inside of turbulence grids.
- New FieldVelocityWeight parameter, which controls how strongly the field velocity affects particles.
- Support convex mesh collision with turbulence grids.
- Support convex mesh collision with basic IOS.
- Clearer emitter debug visualization hints.
- New semantics for particle rendering: COLOR_FLOAT4 and generic 32-bit USER_DATA.
- All turbulence grid parameters now exposed through NxParameterized interface.
- Temperature visualization now shows different colors for temperatures.

Fixes

- Fixed collision issues with turbulence grids which used to cause unrealistic rigid body interaction.
- Turbulence actor LOD is now properly disabled when the module’s LOD is disabled.
- Fixed crash issue in turbulence simulation.

APEX Particles 1.2

Removed

- Particles are only supported on Windows.
- maxObjectCount is no longer an IOFX property
- Most NxFluidIOSActor methods have been removed, this class is not exposed at runtime.
- NxModuleIofx::createRenderableIterator() removed. IOFX Actors are now only reachable via NxApexRenderVolumes.
- NxModuleIofx::getIofxTypeName() removed. Use NX_IOFX_AUTHORING_TYPE_NAME from NxIofxAsset.h

New Features

- NxApexRenderVolume - this new class adds the ability to partition particle world space in many practical ways. See the IOFX programmers guide for details.

Improvements

- Render resource optimizations. Only one writeBuffer() call is made per fluid simulation (IOS actor) per frame, no matter how many IOFX assets or volumes in use.
- Deferred IOFX actor creation keeps render resource churn and renderable counts to reasonable levels.
- IOFX CUDA and CPU performance should be roughly linear to the number of particles in simulation. The number of IOFX assets should have a limited effect.
- More robust CUDA interop with writeBuffer() fallbacks for frames where mapped buffers are not available.
- Interop can be restricted to double buffering if certain guarantees are met by the game rendering thread.

- Improved level of detail particle culling.

APEX Destruction Release Notes

APEX Destruction 1.3.4

New Features

- DestructibleActor::acquirePhysXActorBuffer uses a new flag in DestructiblePhysXActorQueryFlags. ‘AllowActorsNotInScenes’ (default cleared) will return actors that have not yet been added to a PhysX scene.
- DestructiblePhysXActorQueryFlags::All has changed to DestructiblePhysXActorQueryFlags::AllStates, since other (non-state) values are now in the enum.

Removed

Optimizations

Fixes

Miscellaneous API changes

- DestructibleActor::acquirePhysXActorBuffer has a new function signature. The ‘eliminateRedundantActors’ bool is removed, and the flag ‘AllowRedundancy’ added to DestructiblePhysXActorQueryFlags. These have opposite meanings, and their default values are accordingly opposite.

Known Issues

APEX Destruction 1.3.3

New Features

- Destructible assets authored with imported multiple-mesh fbx files now clip their collision hulls to prevent initial penetration when chunks are created.
- New NxPhysX3DescTemplate interface to replace the undefined PhysX3DescTemplate used in public functions NxApexActorSource::setPhysX3Template and ::getPhysX3Template. To use, create an instance using NxApexActorSource::createPhysX3DescTemplate(), and use NxPhysX3DescTemplate::release() when done.

Removed

- Deprecated functions createTwoWayRb, addTwoWayRb, and releaseTwoWayRb have been removed.

Optimizations

Fixes

- Fixed a bug where destructibles fractured when touching kinematics, even if impact damage was disabled.
- Fixed a bug which may have led to errors when disabling detailed overlap tests with performDetailedOverlapTestForExtendedStructures.

Miscellaneous API changes

Known Issues

- Known issue with improper reference counting when using setSkinnedOverrideMaterial, which could lead to APEX trying to release a resource that hasn’t been requested. Use setResource on the material to manually increment its reference count as a workaround.
- Destructible scatter meshes may create render resources outside of updateRenderResources. If you use scatter meshes your callback must be thread safe.
- Potential threading issue when using instanced rendering with destructibles. A destructible could be rendered twice as a result.

APEX Destruction 1.3.2

New Features

- NxDestructibleParameters now has two new fields: legacyChunkBoundsTestSetting and legacyDamageRadiusSpreadSetting. These control whether or not the legacy settings are used per-actor, with per-asset defaults since it is a part of NxDestructibleParameters. These correspond to the NxModuleDestructible functions setUseLegacyChunkBoundsTesting and setUseLegacyDamageRadiusSpread. The per-actor values may override the module values. If the per-actor values are negative (their defaults are -1), then the module values are used. If a per-actor value is 0, then the corresponding setting is “false” (legacy setting is not used). Otherwise, the legacy setting is used.
- NxUserChunkReport has a new virtual callback function onStateChangeNotify that needs to be added to any user implementations. It gets called when chunk visibility changes occur, if the user has enabled it via NxModuleDestructible::scheduleChunkStateEventCallback.
- Destructible authoring API has new functionality that allows it to import full destruction hierarchies. One does this by passing in an array of chunk parent indices to NxFractureTools::buildExplicitHierarchicalMesh or NxDestructibleAssetAuthoring::setRootMesh. PhysXLab uses this when importing a multi-mesh FBX file. If the chunk hierarchy will reflect the parent/child relationship given by the FBX scene graph.
- NxUserChunkReport has a new virtual callback function releaseOnNoChunksVisible. When the last chunk of an NxDestructibleActor disappears, this callback is called. If the user returns true, APEX will release the destructible actor. If the user chooses to release the actor themselves, they must wait until after fetchResults() completes.

- Voronoi fracture distributes its N sites across all pieces of a multi-mesh.
- Added new FractureVoronoi type to RT fracturing.
- Added maxDepentrationVelocity to behavior groups.
- Fracture event callback runs before PhysX simulate step, so the application has a chance to modify PhysX state before simulation.
- Enabled alpha channel for scatter meshes.
- Added user-defined deletion bounding boxes for destructible chunks. See NxApexScene::addBoundingBox and the deleteChunksLeavingUserDefinedBB and deleteChunksEnteringUserDefinedBB parameters of destructible actors.
- Added customizable velocityIterationCount parameter to destructible actors.
- Better conversion of legacy damage spread parameters to the new behavior. See NxDestructibleParameters::legacyChunkBoundsTestSetting and NxDestructibleParameters::legacyDamageRadiusSpreadSetting.
- User-defined support graphs:
    - NxDestructibleAssetCookingDesc::supportGraphEdges
    - NxDestructibleAssetCookingDesc::supportGraphEdgeCount
    - NxDestructibleAssetAuthoring::cacheChunkOverlapsUpToDepth
    - NxDestructibleAssetAuthoring::clearChunkOverlaps
    - NxDestructibleAssetAuthoring::addChunkOverlaps
    - NxDestructibleAssetAuthoring::removeChunkOverlaps
    - NxDestructibleAssetAuthoring::getCachedOverlapCountAtDepth
    - NxDestructibleAssetAuthoring::getCachedOverlapsAtDepth.

Removed

- Support for GPU rigid bodies has been removed.
- Functions createTwoWayRb, addTwoWayRb, and releaseTwoWayRb have been deprecated.
- Removed ‘enabled’ parameter from

NxDestructibleAssetAuthoring::setChunkOverlapsCacheDepth(bool enabled, physx::PxI32 depth = -1), it is now a function parameter of cookChunks

Optimizations

- By default, NxDestructibleActor::applyDamage() will only raycast against static chunks. To match the behavior of previous releases, pass NxDestructibleActorRaycastFlags::AllChunks to NxModuleDestructible::setDamageApplicationRaycastFlags(). You can also have it raycast against no chunks.
- Multiple optimizations to destructible actor creation.
- Faster applyDamage.
- Disabled internal benefit calculations when not using the sort by benefit feature.
- Improved damage event report performance.
- Improved support graph rebuild/update performance.
- Store only a single cooking scale in the destructible module cached data, and use PhysX 3.x's scaling feature instead. Reduces size of data needed for an application when using destructibles at multiple scales. Note that using a scale of (1,1,1) may still give better PhysX performance.
- Improved render update processing which was previously running on some actors that were sleeping.
- Added ability to disable detailed overlap testing for creation of extended structures. See performDetailedOverlapTestForExtendedStructures destructible actor parameter.

Fixes

- No more render resource allocation in fetchResults when instanced chunks are used.
- Chunks with shapes that had their simulation flags reset

(eSIMULATION_SHAPE) could get their inertia tensors re-calulated incorrectly, leading to bizarre physical behavior.

- Render resource creation for scatter meshes is now happening in updateRenderResources of the DestructibleRenderable, not at Asset creation anymore.
- Fixed case in extended structures where some shapes on broken off islands did not get the correct shape template (including userData).
- Fixed crash in NxDestructibleAsset::releaseAndReturnNxParameterizedInterface when it is called on an asset with existing actors.
- Fixed bug that caused the last chunks' graphics to remain when they were destroyed due to a debris setting.
- Fixed behavior when "keepVisibleBonesPacked" is false.
- Fixed NxDestructibleActorJoint not working correctly when jointed to a PxActor.
- When using RT fracture, the renderable bounds now take the RT-fracture chunks into account.
- Fixed various small issues with destructible graphics updates.
- Correctly set applied damage user data in damage event reports.
- Fixed a possible crash when a destructible asset name is NULL.
- Added scene locks to protect potentially unsafe asynchronous sweeps.
- Fixed possible nuisance assertions on destructible actor release and when creating kinematic non-static destructibles.
- Fixed damage event assertion.
- APEX was intercepting onTrigger callbacks from PhysX and not passing them along to the application.
- Apply impact forces to newly created actors in the scene.
- Fixed a memory leak on PS4.
- Fixed improperly inflated convexes when skinWidth is 0, causing bad collision geometry.
- Fixed issue where destructible bounds kept growing.
- Fixed other miscellaneous crash issues.

Miscellaneous API changes

- Moved rebuildCollisionGeometry from NxDestructibleAssetAuthoring to NxDestructibleAsset.
- Renamed “getInstancedChunkCount” to more accurate name “getInstancedChunkMeshCount”
- API cleanup:
  - ExplicitHierarchicalMesh.h renamed to NxExplicitHierarchicalMesh.h
  - FractureTools.h renamed to NxFractureToolsStructs.h
  - Added NxFractureTools.h to expose helper functions that so far were only accessible through NxDestructibleAssetAuthoring
  - Access NxFractureTools through NxModuleDestructible::getFractureTools()
  - Renamed classes and structs in NxExplicitHierarchicalMesh and NxFractureToolsStructs.h to have an Nx prefix
    - IDisplacementMapVolume -> NxDisplacementMapVolume
    - IExplicitHierarchicalMesh -> NxExplicitHierarchicalMesh
    - IEmbedding -> NxEmbedding
    - IConvexHull -> NxConvexHull
    - MeshProcessingParameters -> NxMeshProcessingParameters
    - NoiseParameters -> NxNoiseParameters
    - SliceParameters -> NxSliceParameters
    - FractureSliceDesc -> NxFractureSliceDesc
    - CutoutParameters -> NxCutoutParameters
    - FractureCutoutDesc -> NxFractureCutoutDesc
    - FractureVoronoiDesc -> NxFractureVoronoiDesc
  - Removed “m” prefix of public members of NxMeshProcessingParameters
    - mIslandGeneration -> islandGeneration
    - mRemoveTJunctions -> removeTJunctions
    - mMicrogridSize -> microgridSize
    - mVerbosity -> verbosity

Known Issues

- Known issue with improper reference counting when using setSkinnedOverrideMaterial, which could lead to APEX trying to release a resource that hasn’t been requested. Use setResource on the material to manually increment its reference count as a workaround.
- Destructible scatter meshes may create render resources outside of updateRenderResources. If you use scatter meshes your callback must be thread safe.
- Potential threading issue when using instanced rendering with destructibles. A destructible could be rendered twice as a result.

APEX Destruction 1.3.1

New Features

- Fracturing now handles open meshes. No API change.
- Performance improvement in fetchResults: Only update poses of awake actors.
- Added doNotCreateRenderable parameter to destructible actors. Optimization if the APEX rendering API is being bypassed.
- Added access to scatter meshes from NxDestructibleAsset.
- NxRenderMeshActor::setMaxInstanceCount: Change the max instance count in the case that the instance buffer was changed.
- Improvement to acquirePhysXActorBuffer: Before, if “eliminateRedundantActors” was true, this function could return multiple copies of a PhysX actor per NxDestructibleActor. In 1.3.1, it will only return one PhysX actor per NxDestructibleActor, at most. Also, this query has been made more efficient.

Fixes

- Fixed crash when two or more structures are joined at DestructibleActor creation.
- Fix for assertion in destructible stress solver sample.
- Fixed destructible module cached data in release builds.
- Properly take into account the bias field on weights in destructible benefit calculation.
- Fixed a bad pointer dereference crash.
- Fixed real-time fracturing thread-safety issues.
- Fixed real-time fracturing override of the PxSimulationEventCallback.
- Fixed applyTransformation to also transform scatter meshes.
- Fixed NxDestructibleActorJoint not working correctly when jointed to the world (one actor NULL).
- Fixed potential crash when PhysX 3.x shape creation fails.

- Fixed bad destructible actor deletion/creation accounting, which could potentially have led to bad reference counts on destructibles.
- Fixed crash: APEX could access deleted actors if actor creation rate or fractureBufferProcessRates are exceeded.
- Fixed an unsafe destructible scene release when using GRB.
- Fixed incorrect rendering when a destructible actor is removed from a structure.
- Fixed an issue where the number of dynamic chunks, reported in the DynamicDestructibleChunkIslandCount stat, could exceed the limit set by calling setMaxDynamicChunkIslandCount.
- In some cases when using instanced mesh rendering, creating a destructible actor could lead to a releaseRenderResources callback. That no longer happens.
- FractureTool now handles multi-fbx import
- FractureTool -x option now works correctly

Known Issues

- Known issue with improper reference counting when using setSkinnedOverrideMaterial, which could lead to APEX trying to release a resource that hasn’t been requested. Use setResource on the material to manually increment its reference count as a workaround.
- Destructible scatter meshes may create render resources outside of updateRenderResources. If you use scatter meshes your callback must be thread safe.
- Potential threading issue when using instanced rendering with destructibles. A destructible could be rendered twice as a result.

APEX Destruction 1.3

New Features

- Behavior groups
  - Some common parameters, such as damage threshold, damage spread, density, etc., are now contained in “Behavior Groups.” Every chunk references a behavior group by index, allowing the user to customize behaviors for different chunks.
- Render proxies for destructibles. The rendering of destructibles is managed by a new object that is independent of the destructible actor. By default you will not see a change, but you may detach this object from the destructible, meaning that the render data will not get deleted when the destructible is deleted. You may delete the renderable when you’re done with it. This is useful for multi-threaded renderers which may have the render data queued up even after the destructible is deleted in the main thread.
- New damage detection (can be reverted to legacy behavior):
  - Exact chunk collision volumes used for hit testing (point and radius damage). This gives better consistency when applying damage to destructibles at different LODs.
- New damage spread
  - damageToRadius is no longer used for point and radius damage. Instead, behavior groups contain a DamageSpreadFunction struct, which contains a minimumRadius, radiusMultiplier, and falloffExponent. It works as follows. When damage is applied, it comes with a radius. That radius is zero for a point damage. In any case, we multiply that damage radius by radiusMultiplier, and then add minimumRadius. This value becomes what we call “maximumRadius.” Then, when applying the damage, every chunk up to minumumRadius takes the full damage, and chunks past the maximumRadius take zero damage. How the damage falls off between the minimumRadius and maximumRadius is

determined by falloffExponent. Basically, a linear function which goes from 1.0 down to 0.0 between min to max is raised to the power falloffExponent, and the result at a given chunk's radius is multiplied by the damage, to get the applied damage.
    - damageToRadius is still used for impact damage, but it is no longer scaled by the size of the destructible. To recover the old behavior, multiply the damageToRadius by the approximate radius of the destructible
- Damage vertex coloring
    - The behavior groups now contain a DamageSpreadFunction for damage coloring, as well as a damageColorChange parameter (Vec4). If damageColorChange != (0,0,0,0), a per-actor color channel is created for the destructible. The initial (asset-supplied) color channel is used to initialize it, or all zeros if none exists. Then, using a similar radial behavior to that used for damage spread (above), the color channel is modified by the damageColorChange vector. This allows for some nice effects using multiple-texture shaders. One difference in the radial function: the falloff is calculated between 0 and max radius, instead of min radius and max radius.
- Ability to specify collision volume properties per chunk depth in PhysXLab
- Damage depth limit: You can specify how many hierarchy depths deep fracturing may occur, relative to the chunk that takes the damage.
- More robust fracturing in PhysXLab. BSPs are stored and fractured at a normalized scale, with the output mesh scaled back to the appropriate world coordinates. The result is that many inexplicable errors (BSPs which looked perfect were generating holes and extra triangles) are now gone.
- Better chunk deletion probability behavior. Debris chunks may be deleted with a user-supplied probability. Now the probability distribution is scaled by the damage taken by each chunk, and normalized. The result is that more chunks disappear near the damage point.

- Limited real-time fracturing. Chunks with no child chunks can be real-time fractured now. This means that a fracture pattern is applied to the chunk at a position specified at runtime. This may be applied recursively. In 1.3, only a glass fracture pattern is available.
- A physical stress solver is available when using PhysX 3.3. Stresses are updated as a destructible structure is fractured, and when they exceed a user-specified limit, fracturing will occur at the highest stress points automatically.
- Scatter meshes
    - Ability to author instanced meshes, randomly scattered about the surface of fractured chunks, rotated and scaled with respect to the surface normal within a range specified by the user. Authoring with preview available in PhysXLab.
- Graphical noise on chunk faces with Voronoi fracturing.
- Faster queries (rayCast and obbSweep) for static chunks, if static-only query flags are selected.
- Exposed several functions to get per-chunk state.

Known Issues

- Known issue with improper reference counting when using setSkinnedOverrideMaterial, which could lead to APEX trying to release a resource that hasn’t been requested. Use setResource on the material to manually increment its reference count as a workaround.

APEX Destruction 1.2.4

Fixes

- Fixed contacts not getting detected between PhysX bodies and destructibles when using GRB.

APEX Destruction 1.2.3

New Features

- Added actor synchronization filtering by damage event depth and fracture event depth.
- Added NxDestructibleActor::setDeleteFracturedChunks() to tell a destructible actor to delete its fractured chunks instead of simulating them.
- Added NxUserDestructibleSyncHandler::onPreProcessReadBegin() and NxUserDestructibleSyncHandler::onPreProcessReadDone() callbacks.
- Edited NxUserDestructibleSyncHandler::onReadBegin() callback.
- Removed NxUserDestructibleSyncHandler::onSwizzleDone() callback.
- Added NxDestructibleActorSyncState struct.
- Edited NxDestructibleActor::setSyncParams() method.
- Edited NxDestructibleActor::setHitChunkTrackingParams() method.
- Added new NxDestructibleAssetStats: maxHullVertexCount, maxHullFaceCount, chunkWithMaxEdgeCount.
- NxDestructibleAsset::createDestructibleActor() renamed to NxDestructibleAsset::createDestructibleActorFromDeserializedState().
- NxDestructibleActor::getPartTM() renamed to NxDestructibleActor::getChunkTM().
- IExplicitHierarchicalMesh::IConvexHull::reduceHull function removes vertices from a convex hull until the given limits are reached (after cooking).
- Added both the destructible actor pointer and the chunk index to the NxApexPhysX3Interface::setContactReportFlags() callback when PhysX 3.x shapes are created.

APEX Destruction 1.2.2

New Features

- Option to include use of stress solver in destructibles. The stress solver attempts to detect and break off chunks deemed to be overly-strained.
- Introduced a new parameter struct “StrutureSettings” in the destructible actor. Parameters that make up “StructureSettings” affect all actors’ settings structure-wide.
    - New parameters “useStressSolver”, “stressSolverTimeDelay” and “stressSolverMassThreshold” introduced for the stress solver.

Authoring Improvements

- Option to remove all T-junctions from a fractured mesh. This allows post-processing such as deformation to be applied (for example in a DCC tool).
- Ability to control the interior materials used when fracturing selected chunks.
- Ability to add noise to the perimeter faces of cutout chunks.

Bug Fixes

- ApexHelloWorld sample destruction scene on PS3 crash was fixed

APEX Destruction 1.2.1

New Features

- Enabled GRB support with PhysX 3.2.1.
- Support multiple interior materials.
- Added hard sleeping option, which turns chunk islands kinematic when they sleep. They may be turned dynamic again if enough damage is applied.
- New sleepVelocityFrameDecayConstant parameter replaces sleepVelocitySmoothingFactor.

Removed

- Removed surface trace functionality (was used for old dust system).

Authoring Improvements

- Edge face noise in cutout.

Known Issues

- Known crash in ApexHelloWorld sample destruction scene on PS3 with heavy destruction load.

APEX Destruction 1.2

New Features

- FractureTool FractureTool is a new command-line utility for fracturing and exporting meshes. It provides a subset of the fracturing features exposed by PhysXLab, and serves as a convenient way of batch processing meshes and testing new features.
- NxDestructibleActor
  - Serialization support for actor state provided via the actor's NxParameterized interface
  - Deserialization support for actor state provided via NxParameterized constructor argument
- Added setGlobalPose/getGlobalPose to NxDestructibleActor (sets the pose of static chunks only).
- Added impactDamageDefaultDepth to NxDestructibleParameters. Chunks up to this depth will take impact damage, unless an override flag is set (see below).
- TAKE_IMPACT_DAMAGE flag has been changed to OVERRIDE_IMPACT_DAMAGE/OVERRIDE_IMPACT_DAMAGE_V At the given depth, the OVERRIDE_IMPACT_DAMAGE flag tells APEX to use OVERRIDE_IMPACT_DAMAGE_VALUE (true or false) instead of the behavior it would get from impactDamageDefaultDepth (see above).
- New dust and crumble callbacks, using NxUserChunkParticleReport.
- Visibility event buffer for NxDestructibleActors using acquireChunkEventBuffer / releaseChunkEventBuffer.
- Option for second bone buffer for frame-delayed chunk transforms, using keepPreviousFrameBoneBuffer.

Improvements

- LOD performance improvements.

- Better instancing of cutout chunks, allows for UV offset instancing so that texture maps don’t need to tile with the chunk instance tiling. This requires a renderer which uses the new NxRenderInstanceSemantic::UV_OFFSET semantic for the ApexRenderMesh instance buffer.

Authoring Improvements

- Voronoi fracturing mode using NxDestructibleAssetAuthoring::createVoronoiSitesInsideMesh and ::createVoronoiSplitMesh.
- Voronoi cell visualization utility using NxDestructibleAssetAuthoring::visualizeVoronoiCells.
- Several performance optimizations.
- Ability to fracture a single chunk using NxDestructibleAssetAuthoring::hierarchicallySplitChunk.
- Ability to re-calculate a collision hull for a chunk using different settings, or a custom-made hull, and trim the hulls against neighbors to eliminate initial overlap: NxDestructibleAssetAuthoring::rebuildCollisionGeometry and NxDestructibleAssetAuthoring::trimCollisionGeometry.
- Ability to control interior UV mapping direction and offset in slice and cutout mode. See FractureTools::FractureMaterialDesc.
- Utility to deliver a render mesh which displays a typical noisy slice surface for the current settings, using NxDestructibleAssetAuthoring::buildSliceMesh.

Removed

- Deprecated damageToPercentDeformation and deformationPercentLimit in NxDestructibleParameters.

Known Issues

- Known crash in ApexHelloWorld sample destruction scene on PS3 with heavy destruction load.

Bug Fixes

- Better fracture behavior (fewer errors such as missing or extra polygons).

Misc.

- chunkCount()/depthCount() renamed to getChunkCount()/getDepthCount(), for consistency with the rest of the API.
- Destructibe actors that contain GRBs do not send debug visualization data to PhysX Visual Debugger.

APEX Destruction 1.1

New Features

- GPU Rigid Bodies The NxModuleDestructibe has settings to enable GPU Rigid Bodies for destruction. If your hardware supports this feature, large numbers of chunks (in the 1000’s) can be simulated in a fraction of the time taken by a CPU. GRB Rigid Body support requires NVIDIA driver 270.81 or later, PhysX 2.8.4 RC6 or later and a CUDA capable GPU.
- Chunk instancing (When authored to instance) correponding chunks between different destructible actors will be rendered using an instance buffer. This is advantageous if there are many destructible actors which reference the same asset.
- Chunk tiling (When authored to tile) matching chunks created from cutout fracturing will be instanced within the same actor, as well as other destructible actors which reference the same asset. This can drastically reduce the memory size of the asset.
- NxDestructibleActor:

    - LOD setting sets the maximum chunk depth which can be fractured, implementing forcePhysicalLOD interface
    - getChunkLinearVelocity and getChunkAngularVelocity API in the actor
    - Per-actor materials in the actor descriptor
    - Can specify a separate render mesh for static chunks which gets drawn in a single draw call, using the renderStaticChunksSeparately field in the actor descriptor
    - can set a separate set of static materials used by the static mesh if renderStaticChunksSeparately is set
    - minimumFractureDepth added to destructible parameters, to limit the size of the pieces that can be broken free

- NxDestructibleActor:
  - Implements applyTransformation interface to geometrically transform an asset
- Fracture event callback now consolidates chunk information to reduce the number of fracture events reported
- NxActor (chunk island) FIFO can now be sorted by “benefit” (takes into account screen size and age of the chunk), so that less-beneficial chunks are removed first.
- NxShape count limit can be set in addition to NxActor count

Improvements

- Chunk creation can be amortized over many frames
- Fracture processing can be amortized over many frames
- Conforms to new LOD system
- Removed the per-chunk thread locks, for better performance and resource usage
- SimpleDestruction has multiple sample scenes, loaded using the keys 1-7

Bug Fixes

- Damage reports are no longer issued when a destructible is set free using setDynamic()
- Several crash bugs fixed

Authoring Improvements

- Ability to have multiple collision hulls per chunk.
- Can cancel the fracture operation
- “Trim face hulls” option in cutout fracturing
- Ability to use multiple UV channels and color channels from FBX

meshes
- cookChunks does not use the internal ExplicitHierarchicalMesh any longer, only information passed in from the descriptor

Removed

Known Issues

- Sensitivity of instancing. Instanced cutout fracturing is very sensitive to the scale of the cutout map. It must tile perfectly in order to instance all chunks across a mesh. Similiarly, if there is backface noise in cutout fracturing, the grid size must be a multiple of the number of fracture map tiles. If not, the chunk meshes won't be exactly duplicated from tile to tile, so they will not instance.
- UV instancing not supported. Therefore the texture maps on the asset must tile with the fracture map if instancing is used. Othterwise there will be a graphical pop when fracturing first occurs on a cutout face.
- Chunk island separation bug. When a chunk does not physically touch any of its neighbors, it forms a chunk support island. When supportDepth is set to that chunk's depth, and fracturing occurs, that chunk will always break free from the rest of the destructible, and the rest of the destructible breaks free as well. This is very common when there is noise in the slicing surface, and so chunk collision hull trimming is performed in order to prevent hull overlaps.

APEX Destruction 1.0 Beta

New Features

- Ability for objects to pass through destructibles when damage is taken. This uses the new materialStrength parameter.
- Chunk neighbor padding value is exposed, allowing the user to tune how near chunk collision volumes need to be for chunks to be considered neighbors for support calculations.
- Tiling feature for SimpleDestruction (-tile command line argument), allows destructibles to be tiled in the scene. This allows you to see how they form extended structures.

Improvements

- Destructible structures can now contain up to 4 billion chunks (increased from 64k chunks).
- Dynamic chunk islands can contain chunks from more than one destructible, so there is no artificial fracturing at destructible boundaries.
- More robust impact damage.
- Better behavior when LOD budget limit is reached.
- New flags for NxDestructibleActor::rayCast and DestuctibleActor::obbSweep, which allow you to override the ACCURATE_RAYCASTS setting in the asset.

Authoring Improvements

- Better chunk mesh island generation
- Improved cutout fracturing. More robust, no more triangles inside non-convex and better normal/tangent generation on edges of chunks.
- Threshold angle for normal smoothing on cutout fracturing.
- Better spacing calculation of grid used for noisy surface fracturing.

- Better fractured mesh cleaning. Can reduce fractured mesh size significantly (on the order of 50% in some cases).
- Option for periodic backface noise, so that tiled chippable destructibles have no discontinuities between them.
- Option to choose if a core mesh is exported with a destructible or not.
- Option to “transfer” texture map from core mesh to neighboring chunks or not.
- Slicing parameters may be set at each depth in the fracture hierarchy.
- Multiple-part meshes may be fractured. These will become the first stage of fracturing.
- Core meshes may also be made from multiple-part meshes.

Removed

- Replaced NxDestructibleActorDesc and NxDestructiblePreviewDesc with parameterized objects.

Known Issues

- Cutout-fractured tiles (closely packed collision bounds) jitter when fractured.
- Errors in destructible structure calculations - when chunk neighbors initially form an island, the whole destructible may be dislodged when any part of the structure it is in gets damaged.
- Chunk mesh island generation can sometimes produce spurious triangles.
- Open meshes can produce spurious triangles when fractured. Small openings are usually OK.
- Destruction API is not fully buffered.
- PhysXLab can easily runs out of memory on a 32-bit OS, with complex fracturing.
- Support structures don’t perform stress calculations, leading to configurations “hanging by a thread”
- When testing collision of chunks with the static geometry in the PhysX

scene (for support determination), only the AABB of the chunk is used.

NVIDIA(R) APEX(TM) SDK Release Notes

Note: APEX SDK has been deprecated

Contents: