MDL SDK Release Notes

Document version 2021.1.2
05 Apr 2022

Document build number 349500.8766

Table of Contents

1 MDL SDK 2021.1.2, build 349500.8766

1.1 Fixed Bugs

1.1.1 General

- Remove wrong error message about failures to construct MDL file paths when using the module builder.

1.1.2 MDL Compiler and Backends

- Remove invalid optimization in DAG hashing.

2 MDL SDK 2021.1.1, build 349500.8264

2.1 Added and Changed Features

2.1.1 Image File Format Plugins

- Support .rgb extension for textures in the SGI file format.

2.2 Fixed Bugs

2.2.1 General

- Fixed filename extension mismatch when exporting textures referenced from MDL modules. Under certain circumstances, the texture was copied, but got a different filename extension, causing problems importing the MDL module again.
- Fixed creation of function calls of the cast operator if the target type has frequency qualifiers. Similarly, fixed creation of function calls of the ternary operator if the argument types have frequency qualifiers.
- Fixed handling of memory allocation failures in `IImage_api::create_canvas()/create_tile()` methods.
- Also encode the simple name of function definitions. For almost all functions this does not make any change since the simple name is usually an identifier, except for a couple of operators from the builtins module.

2.2.2 MDL Compiler and Backends

- libbsdf: Fixed incorrect child normal orientation usage in `df::(color_)fresnel_layer`, `df::(color_)custom_curve_layer` and `df::(color_)measured_curve_layer` for backside hits.
- HLSL backend: Fixed code generation for scene data access functions inside automatically derived expressions.

3 MDL SDK 2021.1, build 349500.7063

3.1 Added and Changed Features

3.1.1 MDL 1.7 Language Specification

- Removed the draft status of this document.
- Clarified that constants can be declared locally as well, which is working since MDL 1.0.
- Clarified that the `thin_film` modifier applies only to directly connected BSDFs that have a Fresnel term.
- Clarified that the `state::normal()` orientation is facing outward from the volume, with thin-walled materials treated as enclosing an infinitesimally thin volume, such that both sides are facing outward and the normal always points to the observer.

3.1.2 General

- Added support for animated textures.
  - The signature of various methods on `IImage` and `ITexture` has been changed. The frame index has been added as first parameter and the order of uvtile index and mipmap level has been flipped. The default arguments have been removed. The old signatures are still available if `MI_NEURAYLIB_DEPRECATED_12_1` is defined. Methods to query the mapping between frame index and frame number have been added.
  - The method `uvtile_marker_to_string()` on `IMdl_impexp_api` and `IExport_api` has been renamed to `frame_uvtile_marker_to_string()`. It is still available under the old name if `MI_NEURAYLIB_DEPRECATED_12_1` is defined. The method `uvtile_string_to_marker()` on both interfaces has been deprecated without replacement. The last component of `IImage::get_original_filename()` is an alternative (if available), or construct a custom string from scratch.
  - The interface `IMdl_resolved_resource` has been split such that it represents an array of instances of the new interface `IMdl_resolved_resource_element`, where each element corresponds to a texture frame (only one for non-animated textures and other resources).
  - The frame parameter has been added to various method of the runtime interfaces `Texture_handler_vtable` and `Texture_handler_deriv_vtable`. A new member `m_tex_frame` to support the intrinsics `tex::first_frame()` and `tex::last_frame()` has been added.
  - The examples "DF Native", "DXR", and "Execution Native" have been extended accordingly.
- Added an overload of `IMdl_factory::clone()` that allows cloning of an execution context. Useful for local changes to the context options.
- Disabled WEBP export in the FreeImage plugin due to memory access violations.
- Changed `mi::math::gamma_correction()` to include the alpha channel, as it is done already in other places.

3.1.3 MDL Compiler and Backends

- The MDL core compiler can resolve frame sequences using frame markers in resources.

- The JIT backend issues an error message if the user specifies a state module that does not contain all necessary functions.
- The MDL core compiler avoids a redundant call to the entity resolver when importing modules.

3.1.4 MDL SDK examples

- Examples Shared
  - All examples load now the `dds` plugin by default.
- Example DXR
  - Added support for the glTF extension `"KHR_materials_emissive_strength"` to the example renderer and to the MDL support module.

3.2 Fixed Bugs

3.2.1 General

- Fixed gamma value of pink 1x1 default textures.
- Fixed a race condition for accessing global core JIT backend options from different threads, which could have caused overwritten options or a crash, by using the thread local core thread context options instead.

3.2.2 MDL Compiler and Backends

- libbsdf: Fixed incorrect flipping of the shading normal for strongly bumped normals. Note that libbsdf requires that state input shading and geometric agree on sideness (it has been forgiving with respect to that due to this bug).
- libbsdf: Fixed a numerical issue for `df::fresnel_factor()` (for ior == 0).
- libbsdf: Fixed implementation of albedo for `df::tint(reflect, transmit)`.
- Fixed handling of resource sets if used inside MDLE archives.
- Fixed a crash inside the MDL core compiler if a material preset with too many arguments is compiled.

3.2.3 MDL SDK examples

- Example OptiX 7
  - Fixed normal orientation (as libbsdf needs side consistency between geometric and shading normal).

4 MDL SDK 2021.0 Beta 2, build 349500.5555

4.1 Added and Changed Features

4.1.1 General

- Added an implementation variant based on `std::atomic_uint32_t` to `Atom32`. This results in a large speedup on ARM, and on a rather small speedup on Windows.

4.2 Fixed Bugs

4.2.1 MDL Compiler and Backends

- Fixed incorrect normal flip for strongly bumped normal input (libbsdf).

5 MDL SDK 2021.0 Beta 1, build 349500.5279

5.0.2 General

- The module `::core_definitions` requires now MDL 1.6 and contains a new material `surface_falloff`.
- Added support for texture selectors. The selector can be queried with methods on `IImage`, `ITexture`, `IVolume_data`, `IValue_texture`, and `ITarget_code`. For non-volume data textures, the supported selectors are restricted to `"R"`, `"G"`, `"B"`, and `"A"` for now.
- Renamed `IVolume_data::get_channel_name()` to `IVolume_data::get_selector()` for consistency with `IImage` and `ITexture`. The old method is still available if `MI_NEURAYLIB_DEPRECATED_12_1` is defined.
- The signature of `IMdl_factory::create_texture()` has been extended to specify the selector. The old signature is deprecated and still available if `MI_NEURAYLIB_DEPRECATED_12_1` is defined.
- Added `IImage_api::create_canvas_from_reader()` to enable canvas creation directly from a reader (in addition to `create_canvas_from_buffer()`).
- Added the methods `get_pixel_type_for_channel()` and `extract_channel()` to `IImage_api`, which are useful for extracting RGBA channels from existing textures.
- Added a warning to catch some wrong implementations of `IMdl_resolved_module::get_module_name()`.
- Improved resource enumeration on modules. The new method `IModule::get_resource()` returns an `IValue_resource` with all details, including gamma mode and selector. The old methods returning the individual bits are deprecated and still available if `MI_NEURAYLIB_DEPRECATED_12_1` is defined.
- Added overloads of `IValue_factory::create()` that accept a range annotation as argument, and a type and an entire annotation block. This makes it simpler to create values that observe range annotations. Modified `Definition_wrapper` to make use of the latter if a parameter has no default.
- Added `IFunction_call::reset_argument()` which sets an argument back to the parameter default (if present), or an value observing given range annotations, or a default-constructed value.
- Extended `IValue_factory::compare()` and `IExpression_factory::compare()` to support floating point comparisons with an optional epsilon range.
- The materials-are-functions feature is now enabled by default. See the documentation referenced from `IMdl_configuration::set_materials_are_functions()`. All examples have been updated to avoid usage of `IMaterial_definition` completely, and `IMaterial_instance` as much as possible.
- Added an `user_data` option to the `IMdl_execution_context`, which allows the user to pass its own interface pointer around, in particular to the methods of `IMdl_entity_resolver`.
- Improved performance of editing instances of `IMdl_function_call`, in particular for instances with a large set of arguments. Depending on the exact circumstances, this can cut the time for a full edit cycle (create transaction, create argument editor, change argument, destroy argument editor, commit transaction) in half. An additional speedup can be obtained by making use of the additional optional argument of `Argument_editor`.
- Extended `IExpression_factory::compare()` to support deep comparisons of call expressions. Useful to figure out whether an exporter needs to export an argument, or can rely on the corresponding default on the definition.

- Export to EXR takes now the quality parameter into account: a value of 50 or less selects `half` as channel type.
- Added `create_reader()` and `create_writer()` to `IMdl_impexp_api`.
- Added `iimage_plugin.h`, such that customers can write their own image plugins.
- Changed the behavior of `IImage_api::adjust_gamma()` to include the alpha channel. This also affects export operations (if `force_default_gamma` is set) and the MDL texture runtime if derivatives are enabled.

5.0.3 MDL Compiler and Backends

- Added support for libbsdf normal adaption for CUDA and native runtime. The `--an` option for the `df_cuda` and `df_native` examples demonstrates the feature (the dummy implementation does not change the normal, though).
- The libbsdf implementations of the functions `df::diffuse_reflection_bsdf` and `df::diffuse_transmission_bsdf` now compensate for energy loss caused by differences of shading and geometric normal.

5.0.4 MDL Distiller and Baker

- Removed all internal MDL distiller targets except for the `"none"` target.

5.0.5 MDL SDK examples

- Example DF Native
  - Added support for custom texture runtime.
- Example Distilling Unity
  - Renamed metallic map to `"mask_map"` and base color map to `"color_map"`.
  - Do not bake uniform values to textures anymore (except for the mask map and the base map), but output them on the console.
  - Removed parameters from material names in the saved texture filenames.
  - Composed cutout opacity and transparency and stored in the base map alpha channel.
- Example DXR
  - Added a new dependency to the DirectX Shader Compiler, which is optional for Windows SDK 10.0.20348.0 and later.
  - Write a memory dump file in case an application crash occurs.
- Example Python Bindings
  - Added many more interface to the Python bindings, e.g., `IBaker`, `ICanvas`, `ICompiled_material`, `IImage_api`, `IMdl_distiller_api`, `IMdl_Module_builder`, `IPlugin_configuration`, and `ITile`, and extended the examples.

5.1 Fixed Bugs

5.1.1 General

- Fixed a bug in mdlm and i18n if the environment variable `HOME` is not set.
- Fixed a bug in i18n which caused command MDL search paths other than `"SYSTEM"` and `"USER"` to get ignored.
- Fixed `IFactory::clone()` when dealing with untyped arrays.
- Fixed `IMdl_backend::deserialize_target_code()` such that the internal DF texture are no longer missing under certain circumstances.
- Fixed `ICompiled_material::get_hash()` to take the material slots for `surface.emission.mode` and `backface.emission.mode` into account. Added enumerators `SLOT_SURFACE_EMISSION_MODE` and `SLOT_BACKFACE_EMISSION_MODE` to `mi::neuraylib::Material_slot`. Also added `SLOT_FIRST` and `SLOT_LAST` to support easier enumeration over all material slots.
- Fixed a crash with default constructed BSDF measurements during MDL export/MDLE creation.
- Removed `const` modifier on distilled materials such that they can be stored in the database.

5.1.2 MDL Compiler and Backends

- Improved numerical stability in `base::coordinate_projection()`.
- Improved performance of `math::blackbody()` implementation.
- Fixed handling of weak imports for MDL < 1.6 if an external entity resolver is set. The semantic of weak imports is now handled by the core compiler itself, an external entity resolver sees now only absolute or strictly relative module names.

5.1.3 MDL Distiller and Baker

- Fixed crash with certain mrule transformation rules.
- Fixed crash with certain combinations of distribution functions in distilled materials, e.g., `df::directional_factor` in material emissions.

6 MDL SDK 2021.0.4, build 344800.9767

6.1 Fixed Bugs

6.1.1 General

- Fixed a rare case of incorrect handling of user-defined type names for structs and enums when encoded names were enabled.

6.1.2 MDL Compiler and Backends

- Fixed non-deterministic behavior with `sincos` calls.

7 MDL SDK 2021.0.3, build 344800.8726

7.1 Added and Changed Features

7.2 Fixed Bugs

7.2.1 MDL Compiler and Backends

- Apply thin film only if thickness > 0.0 (libbsdf).

7.2.2 MDL Distiller and Baker

- Fixed a bug which could lead to a crash for certain materials.

8 MDL SDK 2021.0.2, build 344800.7839

8.1 Added and Changed Features

8.1.1 MDL SDK examples

- Example DXR
  - Added support for various glTF extensions to the example renderer and the MDL support module. The new supported extensions are: `KHR_texture_transform`, `KHR_materials_transmission`, `KHR_materials_sheen`, `KHR_materials_specular`, `KHR_materials_ior`, and `KHR_materials_volume`.
  - Added support for `state::animation_time()` when the animation mode is enabled in the renderer settings.
  - Added support for volume attenuation to the renderer.
  - Update MaterialX dependency to version 1.38.1.

8.2 Fixed Bugs

8.2.1 MDL Compiler and Backends

- Fixed incorrect BSDF evaluation for `df::sheen_bsdf` with a transmitting `"multiscatter"` BSDF parameter.
- Fixed `df::thin_film` implementation for the case material IOR < thin film IOR (libbsdf).
- Creation of compiled materials fails now (instead of crashing) if some internal error condition is detected.

8.2.2 MDL SDK examples

- Example Distilling Unity
  - Fixed potential memory leak.
- Example DXR
  - Fixed the UV coordinate transformations to match the glTF specification. Now the V-coordinate is simply flipped by applying v = 1.0f - v to improve UDIM support.
  - Metallic-roughness texture lookups are fixed and will now be handled strict to the glTF specification.

9 MDL SDK 2021.0.1, build 344800.4174

9.1 Added and Changed Features

9.1.1 General

- Reduced memory usage for DDS textures with subformat BC7.
- Updated `nvidia::core_definitions` with new functionality.
- Added a new execution context option "warning" to silence compiler warnings or to promote them to errors. See `IMdl_execution_context` for details.
- Disabled warnings for deprecated MDL materials and functions by default.

9.1.2 MDL Compiler and Backends

- Reduced compilation times.

9.1.3 MDL SDK examples

- Example Code Generation
  - Added an option to set number of texture result slots.

9.2 Fixed Bugs

9.2.1 MDL Compiler and Backends

- Fixed handling of `nvidia::baking` annotations.
- Fixed lambda results handling in single-init mode for non-HLSL.
- Fixed uncomputed cosine in `sheen_bsdf`'s multiscatter (was broken for a transmitting multiscatter component, libbsdf).
- Fixed missing `color_weighted_layer()` inside the transmission analysis.
- Fixed thin film factor implementation (libbsdf).

10 MDL SDK 2021, build 344800.2052

10.1 Added and Changed Features

10.1.1 MDL 1.7 Language Specification

- OpenVDB has been added as supported 3D texture format.
- Minimum required versions have been added for OpenEXR, Ptex, and OpenVDB to conform to the VFX Reference Platform CY2021.
- Supported texture selector string values have been added for each supported texture format.
- Texture sequences with a sequence marker have been added to the definition of texture file paths.
- The `auto` placeholder type specifier has been added for variable declarations and function return types.
- The `float` type is required to use a 32-bit representation following the IEEE 754 single precision standard. Floating point operations for `float` and `double` may deviate but shall not interrupt nor terminate processing.
- The `int` type is required to be a 32-bit signed integer in two's complement form with wrap-around operations and without exceptions.
- The restrictions on the texture types, light profile data type, and measured BSDF data type have been removed. They can now be used for local variable types, return types, field types in user defined structures, and element type of arrays.
- A selector string parameter has been added to the `texture_2d` and `texture_3d` constructors to support texture formats with multiple data sets and channels in a file. The `anno::usage` standard annotation can be used on texture parameters to pre-select selector values in an integration.
- The operators `=`, `==`, and `!=` have been added for the texture types, light profile data type, and measured BSDF data type.
- Emission has been added to the volumetric material properties.
- The return type of a material definition can have an annotation.
- The description of various standard annotations, like `in_group` and `ui_order`, mention their wider applicability to more elements in MDL.
- The `usage` standard annotation on materials is recommended to be used on the return type and not the material definition to match the recommendation for function declarations.
- The hyperbolic trigonometric functions `cosh`, `sinh`, and `tanh` have been added to the standard math library.
- The re-interpreting bit-cast functions `float_bits_to_int` and `int_bits_to_float` have been added to the standard math library.
- The offset functions `width_offset`, `height_offset`, and `depth_offset` have been added to give full access to OpenVDB bounding box information.
- The functions `first_frame` and `last_frame` have been added to give access to the texture animation bounds.
- The transform function `grid_to_object_space` has been added to give access to OpenVDB bounding box information in MDL object space.

- A `frame` parameter has been added to the `width`, `height`, `depth`, `width_offset`, `height_offset`, `depth_offset`, and `grid_to_object_space` texture functions to select frames in texture sequences.
- A `frame` parameter has been added to the `texture_2d` and `texture_3d` variants of the `lookup_`*ltype* and `texel_`*ltype* family of texture function overloads to select frames in texture sequences.
- The uniform modifier has been removed from the `tint` parameter of the EDF `tint` modifier.
- The VDF `tint` modifier has been added.
- An overloaded variant of the `directional_factor` modifier has been added for EDFs.
- The `sheen_bsdf` has been changed to a modifier by adding a BSDF `multiscatter` parameter.
- The uniform modifier has been removed from the `weight` field of the `edf_component` and `color_edf_component` structures and the upper limit has been removed for the weight.
- The `color_vdf_component` structure and VDF overloads for the `color_normalized_mix` and `color_clamped_mix` mixing distribution functions have been added.
- The mix distribution functions `unbounded_mix` and `color_unbounded_mix` have been added for BSDF, EDF, and VDF.
- An Appendix F has been added defining MDL search path conventions.

10.1.2 General

- The MDL SDK now also supports the ARM platform on Linux (`aarch64-linux-gnu`).
- A Python binding for the MDL SDK has been added. This binding consists of a shared library generated from the API headers using SWIG, and some additional helper functions for ease of use. In addition, there is a stub Python module to make that shared library accessible from Python code. See "Language Bindings" in the API reference for more information. Examples to demonstrate working with the Python binding are included as well.
- In the API, the array constructor now uses –as all other function definitions– named arguments instead of positional arguments. The previously irrelevant parameter names are now enforced to be `"0"`, `"1"`, and so on.
- Added support for the "target model" compilation mode.
- Added a context option `"remove_dead_parameters"` to control the removal of dead parameter in instances of `ICompiled_materials`. Dead parameters only survive if this options is set (enabled by default). Setting it to `false` produces a compatible argument block layout independent of the target material mode.

10.1.3 MDL Compiler and Backends

- Added support for MDL 1.7.
- Avoid duplicate calls to common code for ternary BSDF operators and for distribution function modifiers to reduce the code size for HLSL after compilation by the DirectXShaderCompiler (libbsdf).

10.1.4 MDL SDK examples

- Reduced the verbosity of the `cmake` configuration output.

10.2 Fixed Bugs

10.2.1 General

- Fixed wrong handling of encoded MDL names for user-defined type names and names with suffix indicating older MDL versions.
- Improved documentation and examples to demonstrate how to set the gamma mode, in particular when generating MDL source code via the module builder or when creating MDLEs.

10.2.2 MDL Compiler and Backends

- Fixed a crash if a reserved keyword is used as a type name.
- Fixed inlining of functions when argument expressions referenced parameters of the caller.
- Improved error reporting on broken MDL archives and MDLEs.
- Check upon archive creation that user-provided manifest keys form a valid identifier.
- Fixed creation of annotations for function variants (sometimes a wrong module name was used).
- Fixed potential crash with re-exported `enable_if()` annotations.

11 MDL SDK 2021 Beta, build 334800.351

11.1 Added and Changed Features

11.1.1 General

- This release changes the naming convention used for the DB elements of modules, material definitions, and function definitions. This is necessary to avoid problems that exist with the old naming scheme when names contain certain meta-characters. Note that this does not only effect DB elements whose names contain special characters, but all material definitions. See `IMdl_configuration::set_encoded_names_enabled()` for details. This feature is enabled by default.
- Added the new interface `IMdl_module_builder` which allows incremental building of new MDL modules, as well as editing of existing MDL modules. This interface allows the definition of new struct and enum types, and of new functions and materials (including variants). It also supports references to previous entities from the same module, explicit control of frequency modifiers for parameters, and parameters without defaults. An entity can also be removed from a module (if that entity is unreferenced).

  The new interface can be obtained from `IMdl_factory::create_module_builder()`. See also the new `create_module` example. The method `IMdl_factory::create_variants()` is deprecated and still available if `MI_NEURAYLIB_DEPRECATED_12_0` is defined.
- The API can be configured to treat materials as if they are simply functions with the return type `material`. This means interfaces like `IFunction_definition` and `IFunction_call` can also be used for materials. See `IMdl_configuration::set_materials_are_functions()` for details. This feature is disabled by default. It will be enabled by default in a future release.
- The new method `IMdl_factory::uniform_analysis()` allows to check the `uniform` property of an expression graph.
- Improved performance for loading of MDL modules, in particular parallel loading of modules, and reloading of modules.
- Added `force_default_gamma` parameter to `IMdl_impexp_api::export_canvas()` and `IExport_api::export_canvas()` to perform an automatic gamma adjustment based on the pixel type chosen for export.
- The implementation of `IFunction_definition::get_thumbnail()` and `IMaterial_definition::get_thumbnail()` has been changed to compute the value lazily on demand.
- The system locale used in methods of `IMdl_i18n_configuration` is restricted to two-letter strings to follow ISO 639-1.
- Reduced lock scope during MDL module loading. This avoids callbacks to the entity resolver and related interfaces while holding this lock.
- Added `get_mdl_parameter_type_name()`, `get_return_type()` and `get_semantic()` on `IMaterial_definition` for consistency with function definition. Likewise, added `get_return_type()` on `IMaterial_instance`.
- Added `IMdl_impexp_api::get_mdl_module_name()` to obtain the MDL module name for a module identified by its file name.
- Added `IType_factory::clone()` for type lists.
- Made log messages from plugins available.

11.1.2 MDL Compiler and Backends

- The MDL compiler warns now if a literal value would loose precision in a implicit (or explicit) conversion.
- Improved half vector computation for custom-curve/measured-curve layering: assume refraction for non-thin-walled materials to loose less energy for non-physical glass materials constructed from separate BRDF and BTDF.
- Protect custom-curve evaluations against cosines > 1 to avoid numerical corner cases.
- Restricted several annotations to real functions (i.e. not materials): `intrinsic()`, `throws()`, `const_expr()`, `noinline()`.
- Slightly improved generated HLSL code: the HLSL optimizer can now fold constructions like `vector3(a.x, a.y, a.z)` into `a`.
- Added support for backend option "use_renderer_adapt_normal" to HLSL backend.
- `IMdl_backend::translate_environment()` accepts now a function that returns a `base::texture_return` layout-compatible type.
- Improved speed of the DAG compiler computing material hashes.
- Added some mathematical identities for math functions in the DAG optimizer.
- Allowed annotation `ui_order()` on functions and materials.
- MDL 1.7 support in libbsdf:
  - Unbounded EDF mix.
  - (Color-)unbounded mix for BSDFs.
  - `df::sheen_bsdf`'s multiscatter parameter.
  - `df::directional_factor` for EDFs.

11.1.3 Image File Format Plugins

- The FreeImage plugin is now based on FreeImage trunk r1859 (fixes (at least) CVE-2019-12211 and CVE-2019-12212).
- Added error message with details if a DDS texture can not be loaded.
- The `dds` plugin has been enhanced to support more DDS subformats, in particular BC4/5/6/7-compressed textures.

11.1.4 MDL Distiller and Baker

- Added support for MDL 1.7 distribution functions to the MDL distiller.
- Refactored the MDL Distiller to a plugin. Using the MDL Distiller requires now that the new plugin library `mdl_distiller.so` or `mdl_distiller.dll` is loaded with the `IPlugin_configuration::load_plugin_library()` function beforehand.

11.1.5 MDL SDK examples

- Updated recommended version numbers and use C++ 17 to meet the requirements for the VFX reference platform 2021.
- Dropped support for Kepler GPUs.
- Example Code Generation
  - Added distilling.
- Example Create Module
  - New example to demonstrate the new interface `IMdl_module_builder`.
- Example DF Native
  - New example to demonstrate the integration of a MDL into a renderer by using the native (CPU) backend.
- Example Traversal
  - Added MDL search path handling and distilling.

11.2 Fixed Bugs

11.2.1 General

- Fixed `IFunction_definition::create_function_call()` for the special case of the array constructor: This function uses positional arguments. Ensure that user-supplied argument names are ignored, and instead `"0"`, `"1"`, and so on, are actually used.
- Fixed the methods `ILink_unit::add_material_path()` and `ILink_unit::add_material_df()` if the path involves a template-like function.
- Fixed checks for cycles in call graphs in `IMaterial_instance::create_compiled_material()`.
- Fixed the warnings about removed support for deprecation macros in `include/mi/neuraylib/version.h` to support MS Visual Studio.
- Removed bogus error message if comparison of MDLEs returns inequality.
- Fixed handling of resources in reloaded modules.
- Fixed module builder such that MDL file paths are used for resources, not plain OS file names.

11.2.2 MDL Compiler and Backends

- Avoid optimizations while adding to link units, making it impossible to select expression paths from certain distilled materials.
- Handle correctly auto-import of types that are used only inside struct types.
- Fixed textures with different gamma modes being reported as one texture in `ITarget_code`, when resource resolving was disabled.
- In some rare cases array constructors of kind `T[](e_1, e_2, ...)` were handled incorrectly when exported into an MDLE file. Fixed now.

- Fixed scope handling for declarations inside `then`/`else` and loop bodies.
- Disabled tangent approximation in `::base::transform_coordinate()` to fix a performance regression.
- Fixed a potential crash when an array size constant is used in a body of a function that gets inlined in the DAG representation.
- Fixed missing `enable_if` conditions inside material presets.
- Fixed auxiliary base normals (libbsdf).
- Fixed handling of first component in unbounded mixers (libbsdf).
- Reduced energy loss of `df::diffuse_transmission_bsdf` as lower layer of `df::fresnel_layer`, `df::custom_curve_layer`, `df::measured_curve_layer` (libbsdf).
- Fixed incorrect contribution of `df::diffuse_transmission_bsdf` (for reflection directions) and `df::diffuse_reflection_bsdf` (for transmission directions) for evaluation with bumped normals.
- Ensure that resources cloned from another module due to a default argument are included into this module's resource table.
- Fixed wrong function names generated for HLSL code (contained sometimes `'.'`).
- Fixed wrong return code when setting options for LLVM-based backends.
- Use non-refracted transmission directions for non-Fresnel curve layering (libbsdf). This restores pre-2020.1.3 behavior.
- Fixed a crash when importing a function with texture-typed default arguments.
- Fixed a crash when `df::tint(color,edf)` was used in some context.
- Fixed computation of derivation info for builtin functions that do not have a declaration.
- Fixed code generation not restoring import tables of modules.
- Fixed calculation of the lambda call result index in PTX backend.
- Fixed wrong `static` storage modifier for functions in HLSL.
- Fixed generated function names for HLSL, no more `"func0"` etc.
- Fixed rare code generation failure accessing the first member of a nested compound type (HLSL).
- Fixed the pdf computation of all microfacet BSDFs in mode `df::scatter_reflect_transmit` to contain the selection probability (libbsdf).

11.2.3 MDL Distiller and Baker

- Fixed `IBaker::bake_texture()` to support all pixel types for GPU baking.

11.2.4 MDL SDK examples

- Fixed compilation with VS 2019 and CUDA 11.

12 MDL SDK 2020.1.3, build 334300.6349

12.1 Added and Changed Features

12.1.1 MDL Compiler and Backends

- The performance of the parallel module loading has been improved.

12.1.2 Image File Format Plugin

- Huge speedup for loading of progressive JPEGs if only the metadata is needed.

12.2 Fixed Bugs

12.2.1 MDL Compiler and Backends

- All error messages of recursively imported modules are now reported in the list of error messages during compilation.
- The use of an imported user defined structure type with a nested structure type in an exported function signature has been fixed.
- The check for incorrect varying function call attachments to uniform parameters has been fixed in the MDL SDK API.
- The function `Mdl_compiled_material::depends_on_uniform_scenedata` has been fixed.
- The custom-curve and measured-curve layering has been improved for non-thin-walled transmissive materials to reduce energy loss for some cases of modelling glass with these components.
- The import of the `::std` module in modules of lower MDL version has been fixed.

13 MDL SDK 2020.1.2, build 334300.5582

13.1 Added and Changed Features

13.1.1 MDL Compiler and Backends

- A new HLSL backend option `use_renderer_adapt_microfacet_roughness` has been added, which allows a renderer to adapt the roughness values provided to microfacet BSDFs right before using them. The prototype of the function the renderer has to provide is `float2 mdl_adapt_microfacet_roughness(Shading_state_material state, float2 roughness_uv)`.
- A new execution context option `ignore_noinline` has been added, which allows to ignore `anno::noinline()` annotations, enabling inlining when creating a compiled material. Previously this happened later when generating code for distribution functions. But optimizing at this time could lead to a changed DAG which may not contain the nodes requested by the user anymore.

13.2 Fixed Bugs

13.2.1 General

- Fixed a crash in the `i18n` tool when accessing module annotations.

13.2.2 MDL Compiler and Backends

- Fixed wrong optimization for ternary operators selecting different vector elements in HLSL always returning the true expression.
- Fixed wrong PTX version used for sm_86.
- In single-init mode, don't let a requested `geometry.normal` expression calculate the normal again.
- Fixed analysis of derivative variants of functions not being recognized as depending on `state::normal()`.
- Reduced number of texture result slots used in generated init functions.
- Do not generate HLSL code containing `min16int` to ensure compatibility to Slang.
- Fixed translation of conversion of an 8-bit to a 32-bit integer for HLSL.

13.2.3 MDL SDK examples

- Example Distilling Unity
  - Added documentation.
  - Added missing PTX target.
  - Fixed crash when accessing null parameter value.
  - Fixed verbosity settings.

14 MDL SDK 2020.1.1, build 334300.4226

14.1 Added and Changed Features

14.1.1 General

- Thumbnail paths are now resolved when they are requested. Before, the resolving was done during the module loading.
- A new backend option `eval_dag_ternary_strictly` has been added, which enables strict evaluation of ternary operators (?:) on the DAG to reduce code size. By default it is enabled.

14.1.2 MDL Compiler and Backends

- Added single-init mode for a set of functions added to a link unit, allowing all these functions to reuse values calculated in the init function and stored in the texture results field of the state struct. To enable this mode, the first path in the target function description list given to `ILink_unit::add_material()` must be `"init"`. (Note: the init function will not be marked as `ITarget_code::DK_BSDF` anymore.)
- Improved generated code of compiled materials and lambda functions to benefit from CSE across arguments of the root node.

14.1.3 MDL SDK examples

- Examples Shared
  - Enabled CMake option for linking MSVC dynamic runtime (/MD) instead of static (/MT) by default.
- Example DXR
  - Updated MaterialX support to incorporate latest changes from the MaterialX github repository (branch 1.3.8).
- Example Code Generation
  - Added `-e` option to specify which expressions to compile.
- Example Distilling Unity
  - New example that illustrates the distillation of MDL materials, showcases the compilation of all material expressions into one link unit, and baking of material sub-expressions to a texture that fits the texture channel layout for the Unity material.

14.2 Fixed Bugs

14.2.1 MDL Compiler and Backends

- Fixed `IFunction_call::get_arguments()` for the array constructor, such that it always uses `"0"`, `"1"`, and so on as argument names.
- Fixed failing MDLE export if the `tex::gamma_mode` type is only referenced by an annotation.
- Fixed storing matrices in texture results taking up all the space without much benefit.
- Fixed failure to add functions to link units when the path involves template-like functions.

15 MDL SDK 2020.1, build 334300.2228

15.1 Added and Changed Features

15.1.1 MDL 1.6 Language Specification

- Hyperlinks have been added to the MDL Specification PDF document.

15.1.2 General

- On Linux, CentOS 7 is now the minimum required version.
- Enabled support for MDL modules whose names contains parentheses, brackets, or commas.
- The interface `IMdl_entity_resolver` has been redesigned. Support for resolving resources has been added.
- The new interface `IMdl_module_transformer` allows to apply certain transformations on MDL modules.
- Various API methods have been added in order to reduce the error-prone parsing of MDL-related names: To retrieve DB names from MDL names use `get_db_module_name()` and `get_db_definition_name()` on `IMdl_factory`. To retrieve parts of the MDL name from the corresponding DB element use `get_mdl_package_component_count()`, `get_mdl_package_component_name()`, and `get_mdl_simple_name()` on `IModule`; `get_mdl_module_name()`, `get_mdl_simple_name()` on `IMaterial_definition`; and `get_mdl_module_name()`, `get_mdl_simple_name()`, and `get_mdl_parameter_type_name()` on `IFunction_definition` and `IAnnotation_definition`.
- Added a new overload of `IModule::get_function_overloads()` that accepts a simple name and an array of parameter type names instead of two strings. This avoids the ambiguity when parsing parentheses and commas. The old overload is deprecated and still available if `MI_NEURAYLIB_DEPRECATED_11_1` is defined.
- Improved recursive MDL module reloading: changed the traversal order from pre-order to post-order traversal, avoid flagging a module as changed if it did not change at all.
- Improved `Definition_wrapper`: the creation of functions calls for template-like MDL functions requires now an actual argument list since the dummy defaults for such functions easily lead to function calls with the wrong types in the signature.
- Added more options to control the generation of compiled materials in class compilation mode: Folding of enum and bool parameters, folding of individual parameters, folding of cutout opacity, and folding of transparent layers.
- Added methods to retrieve the MDL version of modules, and the MDL version when a particular function or material definition was added to (and, if applicable, removed from) the MDL specification.
- Added methods to retrieve the MDL system and user paths.
- The legacy behavior of `df::simple_glossy_bsdf` can now be controlled via the interface `IMdl_configuration`.
- The return type of `IFunction_definition::get_body()` has been changed from `const IExpression_direct_call*` to `const IExpression*`.

15.1.3 MDL Compiler and Backends

- Added support for target code serialization in the HLSL, PTX, and GLSL backends. See the new methods `get_backend_kind()`, `supports_serialization()`, `serialize()`, and `get_argument_layout_count()` on `ITarget_code`, and `IMdl_backend::deserialize_target_code()`. The new context option `"serialize_class_instance_data"` for `ITarget_code::serialize()` controls whether per-instance data or only per-class data is serialized.
- Allow total internal reflection for glossy BSDFs with mode `df::scatter_transmit` (libbsdf).
- When derivatives are enabled, `state::position()` is now derivable. Thus, the `"position"` field of `Shading_state_material_with_derivs` is now a derivative type.
- Added `"meters_per_scene_unit"` field to `Shading_state_material`. It is used, when folding of `state::meters_per_scene_unit()` and `state::scene_units_per_meter()` has been disabled via the new `IMdl_execution_context` `"fold_meters_per_scene_unit"` option.
- Added derivative support for matrices.
- Added derivative support for scene data functions. Requires new texture runtime functions `scene_data_lookup_deriv_float`, `scene_data_lookup_deriv_float2`, `scene_data_lookup_deriv_float3`, `scene_data_lookup_deriv_float4`, and `scene_data_lookup_deriv_color` (see `texture_support_cuda.h` in the MDL SDK examples for the prototypes).
- Added `mi::neuraylib::ICompiled_material::depends_on_uniform_scene_data()` analyzing whether any `scene::data_lookup_uniform_*()` functions are called by a material instance.
- Implemented per function render state usage in `ITarget_code`.
- Avoid reporting deprecated warnings, if current entity is already deprecated.

15.1.4 MDL SDK examples

- Examples Shared
  - Added utility headers for strings, enums, I/O, OS, and MDL specific tasks to be used in the examples. Updated examples to make use of the new utility headers.
  - Added GUI classes to illustrate MDL parameter editing and to unify user interfaces in examples in the future.
- Example DXR
  - Added a new more structured user interface with various new features including the loading of scenes and environments from the menu, the replacement of materials, compilation and parameter folding options, and parameter editing in instance compilation mode.
  - Integrated the MDL browser (if built) for the replacement of a selected material.
  - Added shader caching to improve loading times (has to be enabled with option `--enable_shader_cache`).
- GLTF Support
  - Added `KHR_materials_clearcoat` support, also in Example DXR.
- MDL plugin for Arnold

    - Added a new example to illustrate the integration of MDL into an existing advanced CPU renderer.
- Example Code Generation
    - Added a new example to illustrate HLSL, GLSL, and PTX code generation.
- Example OptiX 7
    - Added a new example to illustrate the use MDL code as OptiX callable programs in a closest hit shader, and alternatively, how to link the MDL code directly into a per-material closest hit shader for better runtime performance.
- Example Native
    - Added missing scene data functions of custom texture runtime.

15.2 Fixed Bugs

15.2.1 General

- Fixed documentation of `Bsdf_evaluate_data` structs: eval function results are output-only, not input/output.
- Fixed compilation of materials using the array length operator.
- Fixed crash on CentOS 7.1 when importing non-trivial MDL modules.
- Fixed incorrect behavior during function call creation when implicit casts were enabled.

15.2.2 MDL Compiler and Backends

- Fixed file resolution during re-export of MDLE modules.
- Fixed missing clearing of context messages when creating a link unit.
- Fixed detection of absolute file names on Windows for MDLEs on a network share.
- Fixed support for the read-only segment and resources inside function bodies when compiling for the native target.
- Fixed rare crash/memory corruption than could occur on MDLE creation.
- Fixed possible crash when inlining a function containing a `for (i = ...)` loop statement.
- Fixed potential crash in the auto importer when imports of the current module are erroneous.
- Fixed handling of suppressed warnings if notes are attached to them, previously these were attached to other messages.
- Fixed possible crash in generating MDLE when array types are involved.
- Fixed printing of initializers containing sequence expressions, it is `T v = (a,b);`, not `T v = a, b;`.
- Improved AST optimizer:
    - Write optimized `if` conditions back.
    - Write optimized sub-expressions of binary expressions back.

    - Handle `constant && x`, `constant || x`, `x && constant`, `x || constant`.
- Fixed folding of calls to `state::meters_per_scene_unit()` and `state::scene_units_per_meter()` in non-inlined functions.
- Fixed wrong code generation for int to float conversions with derivatives.
- Fixed a bug in the generated HLSL code that caused wrong calculations because loads were erroneously placed after calls modifying memory.
- Fixed checking of valid MDL identifiers (names starting with `"do"` were treated as keywords, but not `"do"` itself).
- Fixed overload resolution for MDL operators.
- Fixed crash in MDL runtime when using nonexistent image files with MDL.
- Fixed invalid translation of `int` to `float` conversion with derivatives enabled.
- Fixed broken `math::sincos()` on vectors.
- Fixed failing MDLE creation due to several missing or non-exported entities (constants, annotations).
- Fixed failing MDLE creation if the main module was < MDL 1.6, but imported an MDL 1.6 module.
- Fixed failing MDLE creation if non-absolute imports of `::base` were used.
- Fixed rare crashes occurring when the array constructor is used in annotations.
- Fixed lost enumeration of BSDF data textures used by the libbsdf multiscatter.

15.2.3 MDL SDK examples

- Examples Shared
    - Fixed failing CUDA checks when minimizing application.

16 MDL SDK 2020.0.2, build 327300.6313

16.1 Added and Changed Features

16.1.1 MDL Compiler and Backends

- Reduced the minimum roughness threshold for microfacet BSDFs from 1e-3 to 1e-7 to make them usable for mirrors and clear glass, which is inefficient but could be required by ubershaders.
- Added `"ro_data_segment"` field to `Shading_state_environment` (`"ro_data_segment_offset"` for HLSL).
- Use `"direction"` for the field name of `Shading_state_environment` (HLSL only).
- Made `state::position()` derivable.

16.2 Fixed Bugs

16.2.1 MDL Compiler and Backends

- Fixed some rare cases were resources inside MDL functions got lost.
- Fixed crash in MDL code generators due to MDL core compiler missing some error messages when a (wrong) member selection has the same name like an enum constant.
- Fixed rare NaN in microfacet sampling.
- Fixed error value of `ITarget_code::get_body_*()` functions.
- Fixed return value of `ITarget_code::create_argument_block()` when required resource callback is missing.
- Fixed read-only data segment data not being set for native lambdas.
- Fixed resource enumeration when compiling multiple expressions in a link unit with `add_material()`: ensure that resources in material bodies are enumerated first.

17 MDL SDK 2020.0.1, build 327300.3640

17.1 Added and Changed Features

17.1.1 General

- On Windows, we recommend Visual Studio 2017 or 2019. Visual Studio 2015 is still the minimum requirement.
- The standalone tools mdlm and i18n have been extended to support Unicode package/module names.

17.1.2 MDL SDK examples

- `example_dxr`:
  - Enhanced dependency tracking when reloading materials to also update indirectly affected materials.
  - Only the compiled material hash is considered to detect reusable generated code which allows to reuse existing materials for structurally equal instances.
  - Skip format conversion for multi-scatter lookup data.
  - Added support larger glTF scenes up to 2GB.
  - Improved cleanup when loading erroneous scenes.

17.2 Fixed Bugs

17.2.1 General

- Fixed handling of resources inside function bodies. Previously, these resources were not found under some conditions, causing black textures for instance.
- Fixed too strict error checks for creation of function calls of the array index operator, the ternary operator, and the cast operator.
- Fixed creation of variants without specifying any annotations where the annotations of the prototype were erroneously copied to the variants.
- Fixed loading of string-based modules with absolute file paths for resources.
- Fixed documentation of generated code interfaces: The results of eval functions are output-only, not in/out.

17.2.2 MDL Compiler and Backends

- Fixed a subtle bug in one of the code caches, which caused ignored argument changes under some complex conditions. Typically, boolean parameters were vulnerable, but could happen to parameters of any type.
- Fixed MDL archive tool failures with Unicode package names. The MDL version of such archives is now automatically set to MDL 1.6 as lowest necessary version.
- A bug in the resource handling was fixed that previously caused resources to be resolved and loaded more that once, possibly leading to failures if search paths had been changed in between.

- Fixed the MDL core compiler's analysis pass. Some analysis info was computed but not annotated, causing JIT failures on functions that consists of a single expression body only.

18 MDL SDK 2020, build 327300.2022

18.1 Added and Changed Features

18.1.1 General

- A new function `mi::neuraylib::IMdl_compiler::get_df_data_texture()` has been added.
- A new function `mi::neuraylib::ITarget_code::get_texture_df_data_kind()` has been added.
- A new enum `mi::neuraylib::Df_data_kind` has been added.
- A new flag on `mi::neuraylib::IMdl_configuration` instructs the MDL compiler to keep the names of let expressions and expose them as temporaries on MDL material and function definitions. This brings the structure of the material/function definition presented in the API closer to the one in the .mdl file.
- The FreeImage plugin is now based on FreeImage 3.18.0.

18.1.2 MDL Compiler and Backends

- Support for the chiang `hair_bsdf` has been added to the code generator for distribution functions.
- Changes to the internal representation of builtin MDL operators. MDL supports a variety of operators, potentially featuring an endless number of instances:
  - array index `operator[]`
  - array length symbol
  - ternary operator `?:`

  Previously, 'local' definitions were created for every used instance of these operators in an MDL module:
  - array index operator on type T in module M: `M::T@(T,int)`
  - array length symbol on type T in module M: `M::T.len(T)`
  - ternary operator on type T in module M: `M::operator?(bool,T,T)`

  This representation had several drawbacks:
  - there might be one definition for the same operator in every module
  - if the operator was not used inside the source of a module, it was not created

  Especially the second point lead to several problems in the editing application. Hence, starting with the 2020.0.0 release, the internal representation was changed and operators are now represented by 'global' template-like definitions:
  - array index operator: `operator[](<0>[],int)`
  - array length operator: `operator_len(<0>[])`
  - ternary operator: `operator?(bool,<0>,<0>)`

  In addition, the name of the cast operator was changed from `operator_cast()` to `operator_cast(<0>)`. Drawback: When inspecting the types of the operators definition, 'int' is returned for the template types, but this might be changed in the future by expanding the type system.
- Support for HLSL scene data renderer runtime functions has been added. See the `scene_data_*` functions in `mdl_renderer_runtime.hlsl` MDL SDK DXR example for an example implementation.

18.1.3 MDL SDK examples

- `example_dxr`:
  - The texture loading pipeline has been simplified and support for UDIM textures has been added.
  - Support for scene data introduced in MDL 1.6 (prim vars) has been added.
- `example_df_cuda`:
  - Support for evaluation of hair-bsdfs on an analytical cylinder has been added.

18.2 Fixed Bugs

18.2.1 MDL Compiler and Backends

- Support for multiple multiscatter textures in one target code object has been fixed.
- Support for multiscatter textures with disabled `resolve_resources` backend option has been fixed.
- Multiple HLSL code generation problems leading to non-compilable code have been fixed.
- Names of member selection operators/field access functions for builtin vector types have been fixed. For example: `float3(float,float,float).x(float3)` is wrong, whereas `float3.x(float3)` is correct. This affects `bool`, `int`, `float`, and `double` vector types.

19 MDL SDK 2019.2, build 325000.1814

19.1 Added and Changed Features

19.1.1 MDL 1.6 Language Specification

- The file path resolution algorithm has been changed to treat weak relative paths the same as strict relative paths if the referring MDL module has an MDL version of 1.6 or higher. Furthermore, the error checks have been simplified to only protect relative paths from referring to files in other search paths.
- The import of standard library modules has been changed in all examples to use absolute path imports.
- An additional way of defining functions has been added using an expression instead of a procedural function body.
- Let-expression can also be applied to functions defined using an expression.
- The limitation has been removed that package names and module names can only be identifiers.
- The new using `alias` declaration has been added to enable the use of Unicode names for module names and package names.
- The description has been clarified that standard module names shadow only modules of the same fully qualified name while modules in subpackages can have a standard module name as their unqualified name.
- The new `scene` standard library module has been added with `data_isvalid`, `data_lookup_ltype`, and `data_lookup_uniform_ltype` functions.
- The new `multiscatter_tint` parameter has been added to all glossy BSDF models to enable energy loss compensation at higher roughness values.
- The new `df::sheen_bsdf` bidirectional scattering distribution function has been added.
- The new `df::tint` modifier overload has been added for the hair bidirectional scattering distribution function.
- The new `df::tint` modifier overload has been added for the separate tinting of the reflective and transmissive light paths of a base BSDF.

19.1.2 General

- The new API functions
  - `mi::neuraylib::IModule::reload()`
  - `mi::neuraylib::IModule::reload_from_string()`
  - `mi::neuraylib::IModule::is_valid()`
  - `mi::neuraylib::IMaterial_definition::is_valid()`
  - `mi::neuraylib::IFunction_definition::is_valid()`
  - `mi::neuraylib::IMaterial_instance::is_valid()`
  - `mi::neuraylib::IMaterial_instance::repair()`
  - `mi::neuraylib::IFunction_call::is_valid()`

  - `mi::neuraylib::IFunction_call::repair()`
  - `mi::neuraylib::ICompiled_material::is_valid()`

  have been added to support reloading of MDL modules.

- The requirements on MDL module names have been relaxed according to the MDL 1.6 Specification to allow loading of modules with Unicode names.
- The new API functions
  - `mi::neuraylib::ITarget_code::get_callable_function_df_handle_count()` and
  - `mi::neuraylib::ITarget_code::get_callable_function_df_handle()`

  have been added.
- The new API function `mi::neuraylib::ITarget_code::get_texture_df_data()` has been added.
- `mi::neuraylib::IMdl_compiler::load_module()` can now be called from multiple threads to allow loading modules in parallel. To support custom thread blocking the new interfaces
  - `mi::neuraylib::IMdl_loading_wait_handle` and
  - `mi::neuraylib::IMdl_loading_wait_handle_factory`

  have been added.
- The new API functions
  - `mi::neuraylib::IMaterial_definition::get_body()`
  - `mi::neuraylib::IMaterial_definition::get_temporary_count()`
  - `mi::neuraylib::IMaterial_definition::get_temporary()`
  - `mi::neuraylib::IFunction_definition::get_body()`
  - `mi::neuraylib::IFunction_definition::get_temporary_count()`
  - `mi::neuraylib::IFunction_definition::get_temporary()`

  have been added.
- The signature of the function `mi::base::ILogger::message()` has been changed.
- The API function `mi::neuraylib::ITransaction::edit()` has been adapted to disallow editing of database elements of type `mi::neuraylib::IMaterial_definition` and `mi::neuraylib::IFunction_definition`.
- Support for multiple occurrence of the same annotation has been added to `mi::neuraylib::Annotation_wrapper`.
- Support for deprecated features guarded by `MI_NEURAYLIB_DEPRECATED_8_1` and `MI_NEURAYLIB_DEPRECATED_9_1` has been removed.

19.1.3 MDL Compiler and Backends

- Support for MDL 1.6 has been added to the MDL core compiler.
- Limited support for MDL 1.6 features has been added to the backends, in particular, the `scene` module is supported, but currently no code is generated for interrogating the renderer, hence always the default value is returned.

- Support for MDL 1.6 has been added to the generated code for distribution functions, that is `df::tint(reflection_tint, transmission_tint)`, `df::sheen_bsdf()`, the `multiscatter_tint` parameter of all BSDFs exposing this in MDL 1.6 (note that this requires that all four dimensions of Bsdf_sample_data::xi are set).
- For evaluating parts of distribution functions that are named by handles, `Bsdf_evaluate_data` and `Bsdf_auxiliary_data` are adapted to select individual handles.
- A new backend option `df_handle_slot_mode` to select how evaluate and auxiliary data is passed between the generated code and the render has been added.
- The `bsdf` field of `Bsdf_evaluate_data` is split into `bsdf_diffuse` and `bsdf_glossy`.
- The `Bsdf_sample_data` structure now requires a 4th uniform random number and returns the handle of the sampled distribution part.
- Inlining of functions containing constant declarations into the DAG has been implemented.
- Support for light-path-expressions in generated code for distribution functions via handles has been added.
- Support for retrieving albedo and normal in generated code for distribution functions via generated auxiliary functions has been added.
- The entity resolver has been sped up for built-in modules in some cases where it is clear that the module can only be read from the MDL root.
- The memory size of the DAG representation has been slightly reduced by internalizing all DAG signatures.
- The DAG representation now uses unsafe math operations, especially `x * 0 = 0` for floating point values.
- The GLSL backend supports version 4.60, now.

19.1.4 MDL Distiller and Baker

- The distiller has been extended to support the new MDL 1.6 BSDF types.

19.1.5 MDL SDK examples

- Example `df_cuda` has been adapted to illustrate how to evaluate parts of the distribution functions named by handles via light path expressions (LPEs).
- All examples have been adapted to support processing of Unicode command line arguments.

19.2 Fixed Bugs

19.2.1 General

- An issue in the light profile parser has been fixed: For IESNA LM-63-2002 files the `ballast-lamp` value incorrectly acted as multiplier for the intensity.

19.2.2 MDL Compiler and Backends

- Several issues in the generated code for distribution functions have been fixed:
  - Bugs in the computation of the pdf and eval functions of `df::ward_geisler_moroder_bsdf` have been fixed.
  - Incorrect pdf computation (for `sample`, `eval`, and `pdf` functions) in `df::ward_geisler_moroder_bsdf` and `df::backscattering_glossy_reflection_bsdf` have been fixed.
  - Fixed a missing re-scale of pseudorandom numbers for v-cavities based masking, leading to biased results for `df::scatter_reflect_transmit`.
  - Only use refraction-based half vector for Fresnel-layering, not for all curve layering operations.
  - Add simple inside/outside material support based on IOR comparison to determine which IOR to override in Fresnel layering. This fixes incorrect rendering when BSDFs of type `df::scatter_reflect` and `df::scatter_transmit` are layered using `df::fresnel_layer`, in particular missing total internal reflection.
- The implementation of `math::isnan()` and `math::isfinite()` has been fixed for vector types.
- Printing of quiet NaNs for HLSL has been fixed.
- A crash in the MDL core compiler that could occur if exported types contain errors in their default initializers has been fixed.
- Wrong function names generated from `debug::assert()` calls when placed after a while loop have been fixed.
- The name of the `anno::deprecated()` parameter has been fixed, it is `description`, not `message`.
- The export of MDL modules containing relative imports has been fixed, access to the imported entities is now generated correctly.

20 MDL SDK 2019.1.4, build 317500.5028

20.1 Added and Changed Features

20.1.1 General

- A new function `mi::neuraylib::IValue_texture::get_gamma()` has been added.
- The new functions
  - `mi::neuraylib::ITarget_code::execute_bsdf_auxiliary()` and
  - `mi::neuraylib::ITarget_code::execute_edf_auxiliary()`

  have been added.
- A new function `mi::neuraylib::ICompiled_material::get_surface_opacity()` has been added.

20.1.2 MDL Compiler and Backends

- Code generation for auxiliary methods has been added on distribution functions for potential use in AI-denoising.
- The spectral color constructor `color(float[<N>],float[N])`, `math::emission_color()`, and `math::blackboby()` are now supported in the JIT backend.
- More optimizations regarding elemental constructors in the DAG representation have been implemented.

20.1.3 MDL SDK examples

- `example_dxr` and `example_df_cuda` have been extended to illustrate the use of auxiliary functions.
- A modified version of `example_dxr` has been added to illustrate the usage of MDL in a multi-threaded context.
- Camera controls have been improved and new options have been added to `example_dxr`.

20.2 Fixed Bugs

20.2.1 MDL Compiler and Backends

- Temporary exponential creation of DAG nodes when using derivatives has been fixed.
- Code generation of parameters reused multiple times in a derivative context has been fixed.
- Relative imports including "." and ".." have been fixed.
- Duplicate global variables in generated HLSL code have been fixed.
- Invalid code generation for HLSL for special materials has been fixed.
- Indeterministic rare compilation errors regarding unknown functions have been fixed.
- Indeterministic rare hangs during compilation with multiple threads have been fixed.

- Under rare condition the code cache could return HLSL code instead of PTX and vice versa. This has been fixed.
- The code cache that was not working under several conditions has been fixed.
- The handling of the `?:` operator on arrays inside the DAG representation has been fixed such that it computes the right name now.
- The handling of unresolved resource paths in the target code has been fixed. Previously all resources were mapped to index 1.

21 MDL SDK 2019.1.3, build 317500.3714

21.1 Added and Changed Features

21.1.1 MDL Compiler and Backends

- Map XOR operators on boolean values to NOT-EQUAL in the HLSL backend to be compatible to SLANG.

21.2 Fixed Bugs

21.2.1 General

- The export of MDLE files from in-memory MDL modules has been fixed.

21.2.2 MDL Compiler and Backends

- A crash in the code generator when handling uniform matrix expressions with automatic derivatives enabled has been fixed.
- A crash in the HLSL code generator for non-default optimization levels has been fixed.
- A crash when adding distribution functions to a link unit after adding non-distribution functions has been fixed.

22 MDL SDK 2019.1.1, build 317500.2554

22.1 Added and Changed Features

22.1.1 General

- A new API class `mi::neuraylib::IAnnotation_definition` has been added.
- The following new functions have been added:
  - `mi::neuraylib::IAnnotation::get_definition()`
  - `mi::neuraylib::IModule::get_annotation_definition_count()`
  - `mi::neuraylib::IModule::get_annotation_definition(Size)`
  - `mi::neuraylib::IModule::get_annotation_definition(const char*)`
- Added a boolean option `fold_ternary_on_df` to fold ternary operators on *df types in material bodies even in class compilation mode: Some uber-materials extensively use the ternary operator to switch bsdfs. This causes a lot of generated code and is not supported by all renderers. Enabling this option will compile all arguments that control the condition of such a ternary operator in. Changing this arguments will then require a recompilation.

22.1.2 MDL SDK examples

- Support for the glTF extension `KHR_materials_pbrSpecularGlossiness` has been added to `example_dxr`.

22.2 Fixed Bugs

22.2.1 General

- Wrong copying of gamma values during cloning of texture values has been fixed. This especially happened for default parameters during material instantiation.
- Fixed problem with dangling user defined types when a module is removed; previously removing a module left all its user defined types alive, preventing creating new (different) ones when the deleted module was reloaded.
- Fixed failure when generating MDLE files that references `intensity_mode` values.

22.2.2 MDL Compiler and Backends

- Fixed wrong optimization of empty do-while loops in the core compiler.
- Fixed imports starting with . or .., these caused wrong package names entered before.
- Fixed printing of float and double constants when MDL code was generated; previously not enough digits were used, causing lost precision when this code was compiled again.
- Fixed a problem where sometimes several uninitialized variables were generated and copied around in generated HLSL code.
- Fixed generation of useless copies, like `t = t;` in generated HLSL code.

- Generated better HLSL code for vector constructor calls, like `floatX(a.x, x.y, ...) => a.xy...`
- JIT compilation of functions involving sin and cos with the same argument on Mac OS has been fixed.
- The implementation of `df::color_weighted_layer` has been fixed.

22.2.3 MDL SDK examples

- `findGLEW` has been fixed in the build script to work with CMake 3.15.

23 MDL SDK 2019.1, build 317500.1752

23.1 Added and Changed Features

23.1.1 MDL 1.5 Language Specification

- The draft status of the document has been removed
- Annotations have been added to annotation declarations.
- A new standard annotation `origin()` has been added, which is used in the MDLE file format to reference the original declarations of refactored elements.

23.1.2 General

- A new function `IMdle_api::get_hash()` has been added.
- A new function `IMdl_compiler::get_module_db_name()` has been added.
- MDLE files now use the new `anno::origin` annotation rather than a custom one.
- The MDLE file format version has been bumped to `1.0`.
- A new interface `mi::neuraylib::IValue_string_localized` has been added.

23.1.3 MDL Compiler and Backends

- The following MDL 1.5 features are now supported by the MDL compiler:
  - `hair_bsdf()` type
  - `df::chiang_hair_bsdf()`
  - `anno::origin` annotation
  - support for annotations on annotation declarations
- The backends were upgraded to use the LLVM 7 library. This means, that the LLVM-IR backend now produces LLVM 7 IR code.
- Allow generating code for `bsdf()` if "compile_constants" options is "on" (default).
- Support for `(color_)measured_curve_layer` has been added to the `HLSL` backend.
- Global constants are only stored in the read-only segment for `HLSL`, when they are larger than 1kB (as for the other backends).

23.1.4 MDL SDK examples

- Support for thin-walled materials has been added to `example_df_cuda` and `example_dxr`.
- Support for relative scene and environment map paths has been added to `example_dxr`.

23.2 Fixed Bugs

23.2.1 General

- Failing MDLE creation when an argument of the MDLE prototype is connected to a function which returns a user-defined type has been fixed.
- A bug leading to different output (and therefore different hashes) when exporting the same MDLE more than once has been fixed.
- A failure (error code -8) when creating presets from functions with user-defined return types has been fixed.
- A failure (error code -8) when creating function presets from MDL modules with versions < 1.3 has been fixed.
- When exporting presets from MDLE definitions or MDL definitions containing calls to MDLE definitions in their arguments, the MDLE code is now inlined into the new module, rather than resulting in invalid MDL.
- The missing `origin` annotation on the main definition of an MDLE file has been added.
- Issues resolving MDLE files on UNC file paths have been fixed.

23.2.2 MDL Compiler and Backends

- JIT compilation of cast operators has been fixed.
- Native code execution on Mac has been fixed.
- Struct member access for MDLE types containing a dot in their file path has been fixed.

24 MDL SDK 2019.1 Beta, build 317500.683

This is a beta release of the new MDL SDK. It is meant for early evaluation and integration prototypes. Do not use this for production code. Please see the list of known bugs and restrictions below.

24.1 Added and Changed Features

24.1.1 MDL 1.5 Language Specification

- A new cast operator has been added to support assignments between structurally equivalent user defined structure types and value equivalent enumeration types to support workflows with the new MDLE format. Beginning with MDL 1.5, `cast` is a reserved word.
- A new field named `hair` of type `hair_bsdf` has been added to the material type, which represents the shading model applicable for hair primitives. Beginning with MDL 1.5, `hair_bsdf` is a reserved word.
- A new elemental distribution function `df::chiang_hair_bsdf` has been added as a hair shading model.
- A new distribution function modifier `df::measured_factor` has been added to support microfacet coloring based on the angle between the half-vector and the shading normal in addition to the angle between the half-vector and the incoming ray direction.
- The new Appendix D – MDLE File Format defines a new container format for a self contained MDL material or function including all of its dependencies and resources.
- The new Appendix E – Internationalization defines the use of XLIFF files for the localization of MDL string annotations.

24.1.2 General

- A new function `IType_factory::is_compatible()` has been added to check if one MDL type can be cast to another.
- A new function `IExpression_factory::create_cast()` has been added.
- A new configuration interface `IMdl_configuration` has been added, which can be used to control the behavior of the SDK regarding the automatic insertion of casts when assigning compatible but different types to arguments of MDL instances.
- The `IMdl_discovery_api` has been extended to also support discovery of resources and XLIFF files.

24.1.3 MDL Compiler and Backends

- A new backend `mi::neuraylib::IMdl_compiler::MB_HLSL` for `HLSL` code generation has been added. Please refer to the `execution_hlsl` and `demo_rtx` examples for examples on how to use it.
- The CUDA/OptiX backend expects some new functions in the user provided renderer runtime to allow using resources unknown at compile-time via argument blocks with class compilation:
  - `bool tex_texture_isvalid(Texture_handler_base const *self, tct_uint texture_idx)`
  - `void tex_resolution_3d(int result[3], Texture_handler_base const *self, tct_uint texture_idx)`

- `bool df_light_profile_isvalid(Texture_handler_base const *self, tct_uint resource_idx)`
- `tct_float df_light_profile_power(Texture_handler_base const *self, tct_uint resource_idx)`
- `tct_float df_light_profile_maximum(Texture_handler_base const *self, tct_uint resource_idx)`
- `bool df_bsdf_measurement_isvalid(Texture_handler_base const *self, tct_uint resource_idx)`

The `tex_resolution_3d()` function fills the width, height and depth for the given texture index into the respective result entry. The other functions are implementations for the corresponding MDL functions. See `examples/mdl_sdk/shared/texture_support_cuda.h` for an example implementation.

- The compiler support for the ternary operator on material and material sub types has been improved. Several materials that caused compile errors before are now compiled flawless.
- The MDL compiler now correctly issues an error when the called object is not a function, detecting (wrong) code like f()().
- The compiler generated now correct MDL code when exporting functions containing a dangling if construct.
- The compiler now (correctly) forbids the use of resource types as parameter types in annotations.
- The PTX backend does not use global counters to generate temporary identifiers anymore, this greatly improves PTX cache hits.

24.1.4 MDL Distiller and Baker

- The Baker can now bake boolean constants right on the GPU.

24.1.5 MDL SDK examples

- A new Direct3D 12 example `demo_rtx` has been added which illustrates how to use the `HLSL` back-end in an RTX-based real-time path tracer.

24.2 Fixed Bugs

24.2.1 General

- Missing imports for user-defined function return types which caused MDLE creation to fail, have been added.
- The handling of conversion operators and constructors when creating MDL AST expressions has been fixed.
- The conversion of array constructors to MDL AST expressions has been fixed.
- The use of implicit conversion functions inside `enable-if` expressions is no longer forbidden.

24.2.2 MDL Compiler and Backends

- Unnecessarily slow code generation for very big output has been fixed.
- Wrong code generation for `df::bsdf_measurement_isvalid()` has been fixed.
- Creating DAG call nodes for ternary operators returning non-builtin types has been fixed.
- For the native and PTX backends, wrong order of array elements returned by `math::modf()` has been fixed.

24.3 Known Restrictions

- The new `hair` field in the `material` structure and the new `hair_bsdf` type from the MDL 1.5 Specification are not yet supported by the MDL compiler.

25 MDL SDK 2019, build 314800.830

25.1 Added and Changed Features

25.1.1 General

- This release contains a preview version of the new MDL encapsulated (MDLE) file format. Please note that MDLE files generated with this SDK version will be invalidated with the next update.
- A new API component `mi::neuraylib::IMdle_api` has been added, which can be used to create MDLE files.
- The function `mi::neuraylib::IMdl_compiler::load_module()` can now load MDLE files. Please note that the option `bool "experimental"` has to be set in the `mi::neuraylib::IMdl_execution_context` in order to enable support for `MDL 1.5`, which is needed for MDLE files.
- The functions
  - `mi::neuraylib::IMaterial_instance::is_default()` and
  - `mi::neuraylib::IFunction_call::is_default()` have been added.
- The standalone tool `mdlm` has been extended with MDLE specific commands.
- The class `mi::neuraylib::IBaker` has been extended to allow baking of constants of type `mi::IBoolean`.

25.1.2 MDL Distiller and Baker

- A new distilling target `transmissive_pbr` has been added.

25.1.3 MDL SDK examples

- A new example `example_mdle` has been added to illustrate the use of MDLE files.
- The example `example_df_cuda` has been adapted to allow loading and rendering of MDLE files.
- A new example `example_generate_mdl_identifier` has been added to illustrate how to generate a valid MDL identifier, e.g., a module name.
- The example `example_distilling` has been extended to support the `transmissive_pbr` distilling target.
- The example `mdl_browser` has been extended to display MDL keywords in the info tooltip as well as above the description in list view mode.

25.2 Fixed Bugs

25.2.1 General

- A bug when translating light profile and bsdf measurement constructors from the MDL SDK API representation to the MDL Core representation has been fixed.

25.2.2 MDL Compiler and Backends

- The hash calculation for struct field access DAG calls for the PTX code cache has been fixed.
- The handling of array parameters in class compilation has been fixed.
- A crash when trying to fold an invalid array constructor has been fixed.
- Missing parentheses when printing operators with the same precedence as MDL or GLSL has been fixed (`"a/(b*c)"` was printed as `"a/b*c"`).
- A potential crash when generating code for distribution functions has been fixed.
- The generated GLSL code for type casts has been fixed to correctly generate `(T)(a+b)` instead of `(T)a + b`.
- An issue with MDL distilling in the case of common subexpressions has been fixed.
- Wrong error messages `"varying call from uniform function"` have been fixed, which were generated by the MDL compiler under rare circumstances for struct declarations.
- Wrong error messages "function preset's return type must be 'uniform T' not 'T'" has been fixed, which were generated by the MDL compiler for function variants if the original function always returns a uniform result but its return type was not declared as uniform T.
- A discrepancy between code execution on CPU and GPU for constant folding of `sqrt(c) (c < 0)` has been fixed. Now `NaN` is computed for both.

26 MDL SDK 2018.1.2, build 312200.1281

26.1 Added and Changed Features

26.1.1 MDL 1.5 Language Specification

- A first pre-release draft of the NVIDIA Material Definition Language 1.5: Appendix E – Internationalization has been added to the documentation set.

26.1.2 General

- Support for the internationalization of MDL string annotations has been added. See the MDL 1.5 Language Specification for details.
- A new API component `mi::neuraylib::IMdl_i18n_configuration` has been added, which can be used to query and change MDL internationalization settings.
- A new standalone tool to create XLIFF files has been added. See `i18n`.
- Calling `mi::neuraylib::ITransaction::remove()` on an MDL module will cause the module and all its definitions and other dependencies to be removed from the database as soon as it is no longer referenced by another module, material instance or function call. The actual removal is triggered by calling `mi::neuraylib::ITransaction::commit()`.
- A new API component `mi::neuraylib::Mdl_compatibility_api` has been added which allows to test archives and modules for compatibility.
- A new standalone tool to manage MDL archives has been added. See `mdlm`.
- A new API class `mi::neuraylib::IMdl_execution_context` intended to pass options to and receive messages from the MDL compiler has been added.
- A new API class `mi::neuraylib::IMessage` intended to propagate MDL compiler and SDK messages has been added.
- A new API function `mi::neuraylib::IMdl_factory::create_execution_context` has been added.
- The signatures of the API functions
  - `mi::neuraylib::IMaterial_instance::create_compiled_material()`
  - `mi::neuraylib::IMdl_compiler::load_module()`
  - `mi::neuraylib::IMdl_compiler::load_module_from_string()`
  - `mi::neuraylib::IMdl_compiler::export_module()`
  - `mi::neuraylib::IMdl_compiler::export_module_to_string()`
  - `mi::neuraylib::IMdl_backend::translate_environment()`
  - `mi::neuraylib::IMdl_backend::translate_material_expression()`
  - `mi::neuraylib::IMdl_backend::translate_material_df()`
  - `mi::neuraylib::IMdl_backend::translate_material()`
  - `mi::neuraylib::IMdl_backend::create_link_unit()`
  - `mi::neuraylib::IMdl_backend::translate_link_unit()`
  - `mi::neuraylib::ILink_unit::add_environment()`
  - `mi::neuraylib::ILink_unit::add_material_expression()`

- `mi::neuraylib::ILink_unit::add_material_df()`
- `mi::neuraylib::ILink_unit::add_material()`

have been changed to use the new class `mi::neuraylib::IMdl_execution_context`. The old versions have been deprecated and prefixed with `deprecated_`. They can be restored to their original names by setting the preprocessor define `MI_NEURAYLIB_DEPRECATED_9_1`.

- The API functions
  - `mi::neuraylib::IMdl_backend::translate_material_expression_uniform_state()`
  - `mi::neuraylib::IMdl_backend::translate_material_expressions()`

  have been deprecated and prefixed with `deprecated_`. They can be restored to their original names by setting the preprocessor define `MI_NEURAYLIB_DEPRECATED_9_1`.
- The utility classes
  - `mi::neuraylib::Definition_wrapper` and
  - `mi::neuraylib::Argument_editor`

  have been extended to provide member access functions.

26.1.3 MDL Compiler and Backends

- Support for automatic derivatives for 2D texture lookups has been added to the PTX, Native x86 and LLVM IR backends. This feature can be enabled via the new backend option `"texture_runtime_with_derivs"`. Please refer to the "Example for Texture Filtering with Automatic Derivatives" documentation for more details.
- Measured EDFs and BSDFs can now be translated to PTX, Native x86 and LLVM IR. Note that the texture runtime needs to be extended with utility functions that enable runtime access to the data.
- Spot EDFs can now be translated to PTX, Native x86 and LLVM IR.

26.1.4 MDL Distiller and Baker

- Distilling speed has been improved by disabling extra error checking in release builds.

26.1.5 MDL SDK examples

- Support for automatic derivatives has been added to the `example_execution_native`, `example_execution_cuda` and `example_df_cuda` examples, which can be enabled via a command line option.
- The `example_execution_native` example has been extended to allow to specify materials on the command line. It is now also possible to enable the user-defined texture runtime via a command line switch.
- The CUDA example texture runtime has been extended with support for measured EDF and BSDF data.
- The MDL Browser is now available as a QT QML Module which can also be integrated in non-qt based applications.
- Initial support for class compiled parameters of type texture, light_profile, and bsdf_measurement has been added to `example_df_cuda`. So far, it is only possible to switch between all loaded resources, new resources cannot be added.

26.2 Fixed Bugs

26.2.1 General

- An error when exporting presets where MDL definitions used in the arguments require a different version than the prototype definition has been fixed.

26.2.2 MDL Compiler and Backends

- A missing check for validity of refracted directions has been added to the generated code for the evaluation of microfacet BSDFs.
- Incorrect code generation for `math::length()` with the atomic types `float` and `double` has been fixed.
- The computation of the minimum correction pattern in the MDL compiler has been fixed.
- The compilation of || and && inside DAG IR has been fixed.
- Pre and post increment/decrement operators when inlined into DAG-IR have been fixed.
- Previously missing mixed vector/atomic versions of `math::min()` and `math::max()` have been added.
- The handling of (wrong) function references inside array constructor and init constructor has been fixed, producing better MDL compiler error messages.
- The hash computation of lambda functions with parameters has been fixed.
- If an absolute file url is given for a module to be resolved AND this module exists in the module cache, the module cache is used to determine its file name. This can speed up file resolution and allows the creation of presets even if the original module is not in the module path anymore.
- A memory leak in the JIT backend has been fixed.
- The generated names of passed expressions for code generation have been fixed.

26.2.3 MDL Distiller and Baker

- A memory leak in the distiller component has been fixed.

26.3 Known Restrictions

- When generating code for distribution functions, the parameter `global_distribution` on spot and measured EDFs is currently ignored and assumed to be false.

27 MDL SDK 2018.1.1, build 307800.2890

27.1 Added and Changed Features

27.1.1 General

- A new API function `mi::neuraylib::ILink_unit::add_material` has been added to translate multiple distribution functions and expressions of a material at once.
- A new API function `mi::neuraylib::IMdl_backend::translate_material` has been added to translate multiple distribution functions and expressions of a material at once.
- A new function `mi::neuraylib::ICompiled_material::get_connected_function_db_name` has been added.
- A new function `IImage_api::create_mipmaps` has been added.
- A new parameter has been added to the functions `mi::neuraylib::ITarget_code::execute*` to allow passing in user-defined texture access functions.

27.1.2 MDL Compiler and Backends

- Diffuse EDFs can now be translated to PTX, Native x86 and LLVM IR.
- Support for passing custom texture access functions has been added to the Native backend. The built-in texture handler can be disabled via the new backend option `"use_builtin_resource_handler"`.

27.1.3 MDL Distiller and Baker

- The distilling of some materials with colored transmission has been improved.

27.1.4 MDL SDK examples

- The `example_df_cuda` example now features simple path tracing inside the sphere to enable rendering of transmitting BSDFs.
- To allow loading of multiple materials within a module, a wildcard suffix `"*"` is now supported in the material name command line parameter of the `example_df_cuda` example.
- The `example_df_cuda` has been updated to illustrate the use of the new function `mi::neuraylib::ILink_unit::add_material`.
- The `example_execution_native` has been extended to illustrate the use of user-defined texture access functions.
- The example_mdl_browser can now be built on Mac OS.

27.2 Fixed Bugs

27.2.1 General

- The handling of archives containing a single module has been fixed in the `mi::neuraylib::IMdl_discovery_api`.
- The handling of relative search paths has been fixed in the `mi::neuraylib::IMdl_discovery_api`.

27.2.2 MDL Compiler and Backends

- Various fixes have been applied to the code generated for BSDF's:
  - The computation of the `evaluate()` function for glossy refraction has been fixed (`df::simple_glossy_bsdf, df::microfacet*`).
  - The `sample()` functions for layering and mixing now properly compute the full PDF including the non-selected components.
  - The implementation of `df::color_clamped_mix()` has been fixed (the PDF was incorrect and BSDFs potentially got skipped).
  - All mixers now properly clamp weights to 0..1.
  - Total internal reflection is now discarded for glossy BSDF (`df::simple_glossy_bsdf`, `df::microfacet*`) with mode `df::scatter_transmit`, as defined in the MDL spec.
- Incorrect code generation for `math::normalize()` with the atomic types `float` and `double` has been fixed.
- The generation of function names for array index functions for modules in packages has been fixed.
- In rare cases, compilation of a df* function could result in undeclared parameter names (missing _param_X error). This has been fixed.
- The compilation of MDL presets of re-exported materials has been fixed.
- In rare cases, the original name of a preset was not computed, which has been fixed.

27.2.3 MDL Distiller and Baker

- Weights in `nvidia::distilling_support::average` are now clamped to fix issues with distilling when layering and mixing weights exceed [0-1] and rely on clamping in the bsdf evaluation.

28 MDL SDK 2018.1, build 307800.1800

28.1 Added and Changed Features

28.1.1 General

- The source code of the MDL SDK is now available on github with an Open Source license. Excluded from the source code release are the MDL distiller, texture baking, and the GLSL backend. An additional lower-level MDL Core API is only available in the source code release.
- The archive tool component now silently ignores extra files in the root directory when MDL archives are created. If this would result in an empty archive (which is forbidden) the ignored files are reported as an error message.
- Access to MDL archives has been accelerated.

28.1.2 MDL Compiler and Backends

- The representation of material instances inside argument expressions has been changed. Previously, material instances inside argument expressions were always inlined, i.e., the material instance was replaced by a call to the material constructor. This made it hard to inspect argument expressions and map them back to user input. Now, a material instance is represented as a call to the material definition, just like any other function call.

28.1.3 MDL SDK examples

- The MDL material examples have been restructured and moved into the examples directory.
- To aid clarity, the MDL SDK examples have been restructured into subdirectories.
- The platform specific Visual Studio solution and Makefiles have been replaced by a CMake build solution.
- Multiple options have been added to the `example_df_cuda` example to make it more useful for comparison with other renderers.

28.2 Fixed Bugs

28.2.1 General

- The IES lightprofile parser has been fixed to allow floating point values for lumen per lamp (note, that the value itself is not used and thus ignored).

28.2.2 MDL Compiler and Backends

- Overload resolution for array constructors in the MDL core compiler has been fixed. Previously, constructs like `T[](S)` could generate errors even if an implicit conversion from type `S` to `T` exists.
- The reported file location by the MDL core compiler for error messages on array constructors has been fixed.
- The MDL core compiler now allows the extra `uv_tile` parameter on tex functions only for MDL 1.4 files. This fixes a bug in the previous release were the compiler accepts them in all language versions.

- A numerical issue with `df::backscattering_glossy_bsdf` and almost identical incoming and outgoing directions has been fixed in libbsdf.
- A numerical issue with `df::simple_glossy_bsdf` and very low roughness has been fixed in libbsdf.

28.3 Known Restrictions

- The CMake build system is not yet supported on OSX.

29 MDL SDK 2018.0.1, build 302800.3323

29.1 Added and Changed Features

29.1.1 General

- A new API component `mi::neuraylib::IMdl_discovery_api` has been added.
- A new API component `mi::neuraylib::IMdl_evaluator_api` has been added.
- The new functions `get_enable_if_conditions()`, `get_enable_if_users()` and `get_enable_if_user()` have been added to the classes `mi::neuraylib::IMaterial_definition` and `mi::neuraylib::IFunction_definition` to ease the handling of `enable_if` - Annotations. Those functions have also been exposed in `mi::neuraylib::Definition_wrapper`.
- A new function `is_parameter_enabled` has been added to the class `mi::neuraylib::Argument_editor`.
- The new functions `get_string_constant_count()` and `get_string_constant()` have been added to the class `mi::neuraylib::ITarget_code`.
- A new function `mi::neuraylib::IMdl_archive_api::get_file()` has been added.

29.1.2 MDL Compiler and Backends

- Compilation time for very simple expressions has been improved.
- An error code when code generation is requested for unsupported BSDFs (`df::measured_bsdf()` is not supported, yet) has been added.
- Diagnostic information for the "`llvm_state_module`" binary option of the CUDA PTX, LLVM IR and native backend in case of errors during compilation has been added.

29.1.3 MDL SDK examples

- A new example `example_distilling_glsl` has been added to illustrate how a material distilled to the UE4-target can be mapped to a GLSL shader.
- A new example `example_discovery` has been added to illustrate the usage of the new API component `mi::neuraylib::IMdl_discovery_api`.
- A new example `mdl_browser` has been added to illustrate how an MDL material browser could look like.
- The examples `example_df_cuda` and `example_df_cuda_gui` have been merged into one.
- Support for boolean, enum and string types has been added to the compiled-material arguments editor of the `example_df_cuda` example.
- A reference implementation of a user-defined LLVM state module (see "`Example for User-Defined State Modules`" in MDL SDK API documentation).

29.2 Fixed Bugs

29.2.1 General

- Some functions of the `mi::neuraylib::Definition_wrapper` were marked as `virtual` by accident. This has been fixed.

29.2.2 MDL Compiler and Backends

- Missing support for `df::color_custom_curve_layer` and `df::color_measured_curve_layer` has been added to libbsdf.
- Variants of materials with default values containing relative resource URLs have been fixed.
- Rare random crashes during JIT compilation for CPU have been fixed.
- Using a material parameter for the gamma mode of texture material parameters has been fixed.
- Compilation of materials with `tex::gamma_mode` parameters has been fixed.
- The auto-import of enum values that was failing when the enum value source was a re-exported function definition has been fixed.
- The missing `mi::neuraylib::Environment_function` function type has been added to `mi/neuraylib/target_code_types.h`.
- The compilation of materials containing calls to functions with deferred size arrays and a deferred size array return type of a different element type (`Int[N] f(float[<N>] a);`) has been fixed.
- The handling of deferred array type expressions inside material bodies of compiled materials has been fixed by using the instantiated types instead of the deferred size arrays.
- The MDL core compiler now explicitly generates an error message if a variant of a function returning a deferred size array type is created, which is forbidden by the MDL specification. Before, this case was forbidden, but the error message was "`undefined symbol`" for the deferred array size symbol.
- The MDL SDK archive API now creates MDL archives in strict mode, i.e. forbids MDL modules that do not strictly follow the MDL spec (but which might be accepted by the MDL SDK compiler in relaxed mode).
- The usage of the ternary operator at the root of a compiled material is now avoided by the MDL compiler. Instead, the operator is moved down to ensure that the root is always a material construction.
- The function remapping feature of the GLSL backend has been enhanced by now allowing to remap functions returning user-defined structs. For every mapped function that is used, the backend creates a define given the MAPPED_<mangled_name> name. This allows to compile only code that is used (in in the case where MDL user types are used it is necessary to remove code that uses structs that are not defined).
- The handling of MDL archive conflicts in the same search root has been fixed: it is an error if a qualified MDL name, or its prefix, can be resolved in more than one archive or package.

29.2.3 MDL Distiller and Baker

- Inconsistent sampling between the CPU and GPU version of the `mi::neuraylib::IBaker` has been fixed.
- Sampling now takes place at the center of the pixel when only one sample per pixel is requested for baking.

29.2.4 MDL SDK examples

- The sphere texture coordinate orientation and tangents have been fixed in `example_df_cuda`.
- The sampling of the environment sphere texture at the poles has been fixed in `example_df_cuda`.
- The importance sampling in `MDL_TEST_MIS_PDF` mode has been fixed in `example_df_cuda`.
- The implementation of wrap modes in the example CUDA texture runtime has been fixed.
- The implementation of `tex::texel_*()` has been fixed in `example_execution_glsl`.

29.3 Known Restrictions

- For the class compilation mode of the native backend, it's not possible to use resources not known at compile-time when generating new target argument blocks.
- The MDL core compiler erroneously accept an additional int2 uv_tile parameter for `tex::width()`, `tex::height()`, and `tex::texel()` function even for MDL < 1.4. This is an error and will be forbidden in future versions of the MDL SDK.

30 MDL SDK 2018.0, build 302800.547

30.1 Added and Changed Features

30.1.1 MDL 1.4 Language Specification

- The list of permitted operators for const-expressions has been extended.
- Support for uv-tilesets including UDIM has been added to the `texture_2d` type and related standard library functions. Corresponding markers have been added to the texture file path definition.
- The new standard annotations `ui_order()`, `enable_if()`, `thumbnail()`, and `usage()` have been added.
- A new standard library `debug` module with `print`, `assert`, and `breakpoint` functions has been added.
- A new distribution function modifier `fresnel_factor` with a complex IOR parameterization has been added.
- New mixing distribution functions with color weights have been added.
- New layerer distribution functions with color weights have been added.
- The type of the `ior` parameter of the `fresnel_layer` BSDF has been changed from `color` to `float`. The new `color_fresnel_layer` BSDF can be used for color IOR.
- Searchability in the MDL Specification PDF document for tokens containing '_' has been improved.

30.1.2 General

- The DSO factory function has been redesigned and renamed to "mi_factory". The old factory function is still available but has been deprecated and renamed to "mi_neuray_factory_deprecated". Please refer to the documentation for details.
- A new API component `mi::neuraylib::IVersion` has been added.
- `mi::neuraylib::INeuray::shutdown()` is now called automatically when the `mi::neuraylib::INeuray` interface is destructed.
- The interface `mi::neuraylib::IModule` has been extended to allow querying the resources defined in a module.
- The interface `mi::neuraylib::IImage` has been extended to support uv-tile image sequences.
- A new function `get_thumbnail()` has been added to the interfaces `mi::neuraylib::IMaterial_definition` and `mi::neuraylib::IFunction_definition`.
- A new class `mi::neuraylib::Annotation_wrapper` with convenience methods for getting and iterating annotations and their values has been added.
- The filtering of parameter names containing '.' in compiled materials when using class-compilation has been removed.
- A new string constant MI_BASE_DLL_FILE_EXT has been added to the API that expands to either ".dll" or ".so" depending on architecture. It has been used to simplify the source code of the examples with regards to shared library loading.
- The dds image plugin has been added to the release for cube-map support.

30.1.3 MDL Compiler and Backends

- Support for MDL 1.4 inside the MDL core compiler has been implemented.
- Support for user defined LLVM implementations of the MDL `state` module has been added to the PTX and LLVM IR backend via the "`llvm_state_module`" binary option of `mi::neuraylib::IMdl_backend::set_option_binary()`. Detailed information can be made available on request.
- Support for generating target code from distribution functions has been added to the PTX, LLVM-IR and native backends. Currently, only BSDFs, BSDF combiners and BSDF modifiers are supported. See `mi::neuraylib::IMdl_backend::add_material_df` and `mi::neuraylib::ILink_unit::add_material_df`.
- The interface `mi::neuraylib::ILink_unit` has been extended to support class compilation.
- The types required to execute JIT compiled native and PTX code have been moved to a new header `target_code_types.h`. With this change, the state structures `mi::neuraylib::Mdl_material_state` and `mi::neuraylib::Mdl_environment_state` have been renamed to `mi::neuraylib::Shading_state_material` and `mi::neuraylib::Shading_state_environment`, respectively. In addition, the matrix pointers in `mi::neuraylib::Shading_state_material` have been changed to float4 pointers to improve compatibility with CUDA and OptiX.
- Methods to retrieve used resources from the generated target code have been added. Resources are represented as identifiers in target code and there must be a way to interrogate the used identifiers and its connection to MDL resources.
- Several optimization inside the MDL core compiler that detect and transform new color variants of *df functions into float variants have been implemented.
- Ambiguous overload error messages in the MDL core compiler have been improved by adding the overload set to the error message notes.
- A check has been added to the compiler that every referenced resource exists (giving warnings in MDL < 1.4 and errors in MDL 1.4 and above). Relative resource paths are transformed into absolute ones.
- Support for "fast" versions of the math runtime functions has been added to the MDL PTX backend. They can be controlled with the fast_math option.
- Built-in min/max instructions and rsqrt() libdevice calls are used in the generated PTX code now.

30.1.4 MDL Distiller and Baker

- The quality of the MDL distiller results has been improved for the diffuse-glossy target in the case of a too dark Fresnel layer, a blend of multiple Fresnel layers, the glossy roughness when blending multiple glossy BSDFs, and materials with thin-walled setting.
- The quality of the MDL distiller results has been improved for the diffuse target in the case of materials with thin-walled setting.
- The distilling result for materials with fresnel layers with an IOR of `0` has been optimized for the UE4 target by removing the layer.

- The quality of the UE4 target has been improved by changing how multiple coats are collapsed in order to better retain specularity.
- The handling of multiple dielectric coatings has been improved for the UE4 target.

30.1.5 MDL SDK examples

- A new example `example_df_cuda` has been added which shows how to use the `mi::neuraylib::ILink_unit` to generate cuda code from MDL material BSDFs and use it in a mini path tracer to render an implicit sphere.
- A new example `example_df_cuda_gui` has been added which shows how to interactively change parameters in the cuda code generated from an MDL material BSDF.
- A new example `example_traversal` has been added which illustrates how to iterate over a `mi::neuraylib::ICompiled_material`.
- The `example_modules` has been extended to list all resources used by a module.

30.2 Fixed Bugs

30.2.1 MDL Compiler and Backends

- A bug regarding missing major version dependency checks for imported MDL modules has been fixed.
- An error in the version number comparison of MDL dependency annotations has been fixed.
- MDL compiler auto import failures for entities with a prefix length > 1, i.e. A::B::X have been fixed.
- The overload resolution for candidate sets with different number of default arguments has been fixed.
- Compiler error messages mentioning the "<ERROR>" type have been fixed.
- Previously missing error messages if a resource named by a strict relative path does not exist are now generated.
- The generation of resource tables for link units has been fixed for all MDL backends.
- For the MDL GLSL backend the generated code for texture lookups has been fixed, a bug regarding missing code when using link unit compilation has been fixed and the generated code for access of invalid resources has been fixed.
- The handling of `from` and `to` parameters in jitted code for `state::transform_scale()` has been fixed.
- A wrong overflow check on decimal literals has been fixed.
- Broken AVX512 support for the native JIT compiler has been disabled.

30.2.2 MDL Distiller and Baker

- Bugs have been fixed that caused the MDL distiller to fail on materials that used a tint modifier on an EDF or mixers on EDFs or VDFs.
- A bug has been fixed that caused the MDL distiller in class-compilation mode to crash on materials that used a conditional operator on a material type or one of the distribution types.
- A bug has been fixed that made the MDL distiller fail on materials that contained a constant `bsdf_component array` or a constant `surface_geometry structure`.
- A bug has been fixed affecting the MDL distiller on materials using the `microfacet_phong_vcavities` BSDF.

30.3 Known Restrictions

- The debug module is not very useful on GPU, yet, due to multithreaded printing onto the console.
- For the class compilation mode of the native backend, it's not possible to use resources not known at compile-time when generating new target argument blocks.

31 MDL SDK 2017.3.1, build 296300.4444

31.1 Added and Changed Features

31.1.1 General

- An error is now returned if mi::neuraylib::set_value fails for enum and resource values.

31.1.2 MDL Compiler and Backends

- Added link mode which allows to generate code for multiple materials in the same compilation unit. This avoids possible problems with duplicate symbols and reduces the total size of code and constant data. See `mi::neuraylib::ILink_unit`.
- Added support for class compilation to the JIT backend. For the PTX backend, the generated code for expressions will require a pointer to the data of the `mi::neuraylib::ITarget_argument_block` of the `mi::neuraylib::ITarget_code` object as an additional parameter. The data can either be manually created by using the information from `mi::neuraylib::ITarget_value_layout` or by using `mi::neuraylib::ITarget_code::create_argument_block()`.
- Added support for enumerating used light profiles and BSDF measurements to `mi::neuraylib::ITarget_code`.

31.1.3 MDL Distiller and Baker

- The quality of the diffuse distilling target has been improved for the case of a too dark Fresnel layer.
- The quality of the diffuse-glossy distilling target has been improved how it handles local normals and how a diffuse and glossy BSDF are mixed together.
- The quality of the UE4/specular glossy distilling targets have been improved by combining multiple glossy lobes with a bias towards lower roughness to keep highlights sharp and by interpreting fresnel/custom curve layered glossy bsdfs with high facing reflectivity as a metallic material. Furthermore, handling of diffuse transmission has been improved which fixes desaturation issues in some transmissive materials.
- The UE4/specular glossy distilling results have been simplified if the same normal is used in multiple layers and for layers/mix components with zero weight.

31.1.4 MDL SDK examples

- Added CUDA and GLSL code execution examples.
- Added Visual Studio 2015 project files.

31.2 Fixed Bugs

31.2.1 MDL Compiler and Backends

- PTX backend: Fixed possible crash in code generated by `mi::neuraylib::IMdl_backend::translate_material_expressions()` due to wrong alignment assumptions of the result pointer.

- Fixed 3D texture access for native jitted code.
- Fixed initialization of arrays of typedef'ed elements.
- Fixed crash with self-referencing initializers.
- Fixed problems with the PTX code cache.
- Fixed constant folding of function calls to simple functions containing assignment operators.

31.2.2 MDL Distiller and Baker

- Thin-walled materials and their respective backface property are now handled correctly in the diffuse and the diffuse-glossy distilling targets.
- A bug distilling nvidia::df::microfacet_phong_vcavities_bsdf has been fixed.
- A bug distilling nvidia::df::ashikhmin_shirley_glossy_bsdf has been fixed.
- A bug when distilling a constant surface_geometry property has been fixed, i.e., if the the normal was set to a literal.
- The handling of 3d textures in the CUDA baker has been fixed.
- The gamma correction of used textures when baking MDL expressions on the GPU has been fixed.
- A GPU memory leak in the baker if CUDA baking is used has been fixed.

31.3 Known Restrictions

- For the class compilation mode of the native backend, it's not possible to use resources not known at compile-time when generating new target argument blocks.

32 MDL SDK 2017.1, build 296300.2288

32.1 Added and Changed Features

- The MDL SDK requires a driver supporting CUDA 9.0, e.g., a driver from the 384 family or higher, if texture baking on a GPU is used. Otherwise, no GPU driver is required. The MDL SDK can generate code for NVIDIA GPUs starting with shader model sm_20 and higher.
- New documentation layout including the reference documentation for the `base.mdl` module.
- Distilling supports compiled materials created with class compilation (flag Compilation_options::CLASS_COMPILATION in function call IMaterial_instance::create_compiled_material).
- The default baker resource in `IMdl_distiller_api::create_baker()` has been changed to `BAKE_ON_CPU`.
- The function `ICompiled_material::depends_on_global_distribution()` has been added.
- Generated PTX code is cached now in order to improve performance when baking the same expression more then once.
- Support for the BSDF in nvidia::df have been added to the MDL distiller targets.

32.2 Fixed Bugs

- Fixed a crash bug in the MDL distiller that could occur when diffuse BSDFs where combined for the `diffuse_glossy` target.
- A bug in the distiller when encountering a `0`-initialized normal has been fixed.
- A bug in the distiller causing materials with mixed EDFs to fail has been fixed.
- Fixed a crash in the MDL compiler which occurred when compiling a material expression containing a call to `float3(float,float,float).y(float3)` created using `IExpression_factory` (bug 18530).
- Return NULL if IValue_factory::create_invalid_df() is called with an argument of a resource type instead of a distribution function type, which fixes a crash bug.

33 MDL SDK 2017.1 Beta, build 296300.1005

33.1 Added and Changed Features

- The new API component `IMdl_distiller_api` has been added to the MDL SDK. It provides functions to transform an arbitrary MDL material into a predefined target model and to bake subexpressions of a material into a texture.
- The API component `IImage_api` has been added to the MDL SDK. Among other useful methods, it provides a default implementation of the `ICanvas` and `ITile` interfaces.
- The interface `IModule` has been extended to allow enumeration of constants as well as struct and enum types declared in an MDL module.
- The interfaces `IType_enum` and `IType_struct` have been extended to provide access to annotations on the type itself as well as for individual enum values and struct fields.
- The methods `set_name()` and `get_name()` have been added to `IValue_enum`. They allow setting and getting the value of an enum by its string representation.
- The deprecated interface `IValue_invalid_ref` and enum value `VK_INVALID_REF` have been removed. Use `IValue_invalid_df` / `VK_INVALID_DF` for references to distribution functions, or the regular resource value interface with `NULL` as name for resources.
- If the methods `reset_file()` or `reset_reader()` fail for any kind of resource, they no longer change the resource in any way.
- The loading of light profiles with non-equidistant angles has been improved. If no explicit resolution is specified in the API call, we attempt to choose a resolution that maintains the angles given in the .ies file as close as possible.
- The FreeImage plugin uses now FreeImage 3.17.0.
- On Linux, plugins are no longer loaded with the `RTLD_DEEPBIND` flag since it causes problems with Address Sanitizer (see `https://github.com/google/sanitizers/issues/611`). The original reasons for that flag are gone, and its absence did not reveal any problems so far.
- A new option `tex_lookup_call_mode` has been added to the `PTX backend` that supports different calling modes for renderer specific texture lookup callbacks. See the API documentation for details.
- `ITarget_code` allows to retrieve the MDL texture shape of the used textures.
- The API now offers a new method `IMdl_backend::translate_material_expressions()` that allows to compile several material sub expressions into one piece of source code.
- `translate_environment()` allows MDL functions that return a `::base::texture_return` instead of a color value.
- Added `ITarget_code::uses_state_parameter()`. This can be used to retrieve the MDL state usage of a compiled material subexpression.
- The SDK now supports a `native` backend which JIT compiles native code for the current CPU. This code can be called directly using the `ITarget_code` interface and exists as long as the interface is valid.
- The MDL compiler now suggests the correction of possibly misspelled entity names inside of error messages.
- Improved PTX code generation for newer GPUs.

33.2 Fixed Bugs

- Fixed start position of postfix expressions in the MDL compiler that leads to wrong positions inside error messages.
- The PTX code generator now correctly generates code for environment functions (using `translate_environment()`).
- Fixed a crash inside the MDL compiler that occurs on rare conditions if a wrong function declaration is processed.

34 MDL SDK 2017.0.1, build 287000.7672

34.1 Fixed Bugs

- Fixed incorrectly issued warning from the MDL compiler if an MDL module imports another MDL module from the same archive.
- Fixed incorrect startup message on Windows (wrong DLL name).

35 MDL SDK 2017, build 287000.5634

35.1 Added and Changed Features

- The interface `IModule` provides now access to types and constants exported by an MDL module.
- The new helper class `mi::base::Log_stream` adapts `mi::base::ILogger` to `std::ostream`.
- The interface `IMdl_compiler` supports now the export of resources.
- The MDL compiler warns now if resources referenced by archive modules are missing or are found outside the archive (bug 17885).
- The MDL compiler simplifies now the path of MDL entities after they were found. This leads to better debug output.
- The computation of hash values used by `ICompiled_material` has been changed. If available, the MDL file path is now used to identify resources. Otherwise, as before, internal identifiers depending on the state of the database are used. This change makes the hash values more consistent across different processes.

35.2 Fixed Bugs

- Fixed endless loop in the MDL compiler that could occur on some wrong MDL input (bug 18001).
- Fixed outdated values returned by `IValue_resource::get_file_path()` if the underlying resource was changed after creation of the value.
- Fixed reported file name of MDL modules loaded from strings (bug 18154).

Retroactively added the following item to the release notes for "MDL SDK 2017 Beta, build 287000.2320":

- The handling of the GLSL backend option `glsl_place_uniforms_into_ssbo` has been fixed.

36 MDL SDK 2017 Beta, build 287000.2320

36.1 Added and Changed Features

- Previously, the MDL compiler allowed implicit conversions from scalar to vector types and from `float` to `color`, contrary to the MDL specification which requires explicit conversions. Now, the compiler emits a warning. In future MDL versions, this will be treated as an error.
- The new methods `IMdl_factory::create_mdl_texture()`, `create_light_profile()`, and `create_bsdf_measurement()` allow to load resources by specifying their absolute MDL file path (as if that string appears inside an MDL module). The new method `IValue_resource::get_file_path()` allows to retrieve the absolute MDL file path.
- If a resource is loaded via an MDL module, the methods `get_original_filename()` on `IImage`, `ILightprofile` and `IBsdf_measurement` return now `NULL` instead of the MDL file path (since there is no original filename available, and the MDL file path is something different). The method `IValue_resource::get_file_path()` provides the MDL file path instead.
- The method `ICanvas::get_layers_size()` has been moved to the base interface `ICanvas_base`.
- The new methods `reset_reader()` on `IImage`, `ILightprofile` and `IBsdf_measurement` allow to load resources from an abstract reader, i.e., resources that do not exist as files on disk.
- The default implementation of `ITile` has been improved to support more than 4 GB of data.
- Support for various fields on `ITexture` that were only relevant for MetaSL has been deprecated. The corresponding methods are still available if `MI_NEURAYLIB_DEPRECATED_8_0` is defined.
- Support for deprecated features guarded by `MI_NEURAYLIB_DEPRECATED_7_1`, `MI_NEURAYLIB_DEPRECATED_NO_EXPLICIT_TRANSACTION`, and `MI_NEURAYLIB_DEPRECATED_NAMESPACE_MI_TRANSITION` has been removed.

36.2 Fixed Bugs

- Fixed the names of the constructor arguments for the builtin types to match the names from the MDL specification (bug 18190).
- Fixed a bug that caused the wrong overload of the `range()` annotation to be selected. For example, `range(float,float)` is now selected for `float p [[ range(1, 2) ]]` , whereas previously incorrectly `range(int,int)` was selected.
- Previously, cycles in material graphs lead to a crash during compilation of the material. Now, when compiling such material the cycle is detected and compilation fails (bug 18078).
- Fixed a crash that could happen if a texture file disappeared between the first access for the meta data and the second access for loading the actual pixel data.
- The handling of the GLSL backend option `glsl_place_uniforms_into_ssbo` has been fixed.

37 MDL SDK 2016.3.1, build 278300.6408

37.1 Added and Changed Features

(none)

37.2 Fixed Bugs

- Fixed comparison of semantic version numbers in the manifest of MDL archives.
- The MDL compiler now issues a warning for modules in MDL archives that import another module outside the archive without declaring that dependency in the `MANIFEST`.
- Fixed detection whether stderr is connected to a console on Windows. Depending on the circumstances this might have lead to missing log output when the default logger was used.

38 MDL SDK 2016.3, build 278300.4305

38.1 Added and Changed Features

- The behavior of `IFunction_call::create_variants()` has been changed. Defaults not explicitly specified are no longer copied from the prototype and made explicit defaults of the variant, but will implicitly use the defaults of the prototype. If such a default of the prototype is later changed, the change also affects the variant. To maintain the previous behavior pass explicit defaults when creating the variants, e.g., start with the defaults of the prototype or the arguments of the material instance/function call instead of with an empty expression list, and then change the defaults as desired.
- The MDL compiler now properly checks the uniform/varying property of material parameter attachments and warns if an expression computing a varying value is attached to an uniform parameter. This warning might turn into an error in future versions.
- The loading of pixel data from textures stored in MDL archives is now done lazily as with textures not stored in MDL archives.
- The new method `IMdl_archive_api::set_extensions_for_compression()` allows to specify the file types to be compressed in an MDL archive.
- The new methods `get_prototype()` on `IFunction_definition` and `IMaterial_definition` provide the name of the prototype for variants.
- The new method `IFunction_definition::is_uniform()` allows to tell apart uniform and varying functions.
- The new method `IMdl_backend::translate_material_expression_uniform_state()` allows to set the uniform state when translating material expressions.
- The GLSL backend avoids now the creation of the ternary operator if the condition is a constant.
- Added support for the MDL `state::transform*()` and `state::object_id()` functions in the GLSL backend.
- The GLSL backend creates now swizzles for MDL expressions of color type.
- All backends use not index `0` to report invalid textures. As a consequence, one invalid texture is reported if at least one texture (valid or not) is used.

38.2 Fixed Bugs

- Fixed a crash that occurred if an MDL function containing a partial write to a struct or vector value like `v.x = expr;` was inlined into a compiled material.
- The MDL exporter no longer generates strict-relative file paths for pre-MDL 1.3 modules.
- Fixed serious slowdown when parsing very long annotations.
- The MDL compiler checks now that the `MANIFEST` of MDL archives is valid.
- Fixed generation of version numbers in the `MANIFEST` of MDL archives.
- Fixed missing execution of the dependency check for MDL modules loaded from MDL archives.
- Fixed the retrieval of `"export.*` keys in the `MANIFEST` of MDL archives that were not reported in all cases.

- Fixed the MDL compiler to accept dates without a not in `MANIFEST` values.
- Fixed parsing of comments in the `MANIFEST` of MDL archives.
- Fixed parsing of UTF-8 characters inside the `MANIFEST` of MDL archives.
- The MDL compiler checks now the rules for character positions inside the `MANIFEST` of MDL archives.
- Fixed parsing of module exports inside the `MANIFEST` of MDL archives that was failing if the module name was shorter than 3 characters.
- Fixed missing error message on Windows when an MDL archive was to be created from a non-existing directory.
- Fixed resolving of strict relative file paths inside of MDL archives.
- The file name reported by `IModule::get_filename()` has been fixed for MDL archives.
- Fixed missing warning message if resolving of MDL file paths fails.
- The maximum line length supported for `.ies` files has been increased to 4096. Proper error handling for lines exceeding this limit has been added.
- Fixed a rare crash if DB elements of type `IFunction_call` are overwritten by function calls with a different return type.
- Fixed creation of global constants in the GLSL backend. Before this change, global constants that were transformed into uniforms or the SSBO buffer could be erroneously generated several times, increasing the amount of necessary memory.

39 MDL SDK 2016.3 Beta, build 278300.573

39.1 Added and Changed Features

- MDL archives as specified in Appendix C of the MDL 1.3 specification are now supported. Using MDL modules is transparent, i.e., it does not matter whether MDL modules (or referenced resources) are located in MDL archives or plain files. The new API component `IMdl_archive_api` allows to create and to extract MDL archives, as well as to inspect their manifest. The new API example `example_archives.cpp` demonstrates these possibilities.
- The method `IMdl_factory::create_variants()` has been extended such that it now allows the creation of annotations with multiple arguments of type `bool`, `int`, `float`, `double`, and `string`.
- Invalid resources are no longer represented by `IValue_invalid_ref` but by the corresponding subclass of `IValue_resource`. Constructing instances of `IValue_invalid_ref` for resource types is deprecated, but still possible and emits a warning. Since the only remaining non-deprecated use of `IValue_invalid_ref` is to represent invalid distribution functions, it has been renamed to `IValue_invalid_df`. The old name is still available if `MI_NEURAYLIB_DEPRECATED_7_3` is defined. Similarly, `VK_INVALID_REF` has been renamed to `VK_INVALID_DF`.
- A new option `"glsl_place_uniforms_into_ssbo"` has been added to the GLSL backend. If the option is set large constant MDL data can be copied into one SSBO object instead of using uniforms, whose size might be very limited.
- The function `unload()` provided with the API examples has been changed: It no longer checks that `dlopen(RTLD_NOLOAD|...)` after `dlclose()` returns `0`. This is not guaranteed by the standard and fails in some circumstances, e.g., unstripped debug builds.

39.2 Fixed Bugs

- Fixed a rare crash in expressions involving the multiplication of a float by a color or vector.

40 MDL SDK 2016.2, build 272800.6312

40.1 Added and Changed Features

(none)

40.2 Fixed Bugs

- The export of compound literals containing enum values has been fixed. Before this fix, the MDL exporter did not qualify enum values that occurred in complex literals (arrays, structs) correctly.
- The MDL exporter no longer generates strict-relative file paths for pre-MDL 1.3 modules.
- Updates to memory-based resources were not always correctly reflected in compiled materials.

41 MDL SDK 2016.2 Beta, build 272800.3649

41.1 Added and Changed Features

- The FreeImage image plugin has been renamed from `freeimage.so/dll` to `nv_freeimage.so/dll` to avoid collisions with the DLL name of FreeImage itself.
- The search paths for MDL modules and resources do no longer accept non-existing directories (and return a new return code in such cases).
- The interfaces `ICall`, `ICall_decl`, and `ITemporary` have been removed. There were only used by the old MDL type system in the context of compiled materials.
- The signature of `IMaterial_instance::create_compiled_material()` has been changed to allow to specify the minimal and maximal wavelength. Previous default arguments have been removed.
- The MDL exporter has been improved in various ways. It supports now memory-based resources, attempts to use the original filename of resources to maintain the meaning, and uses strict-relative file paths to avoid ambiguities.
- The return codes of `IMdl_compiler::export_module()` and `IMdl_compiler::export_module_to_string()` have been changed to allow better diagnosis of error causes.
- The MDL factories have been extended to dump annotations, annotation blocks, and type/value/expression/annotation lists. `IFactory::dump()` has been extended to support an optional transaction parameter.
- The method `ILightprofile::reset_file()` rejects now calls where `resolution_x` or `resolution_y` is 1.

41.2 Fixed Bugs

- Two bugs in the MDL core compiler regarding uniform and non-uniform values have been fixed. The compiler did not correctly check the parameter types of varying functions.

  Hence, errors like this: `T f(uniform T x) varying; varying T g(); f(g())` were not detected. This has been fixed.

  Additionally, functions using varying calls only inside the default parameters are not erroneously marked as varying anymore. Calls inside the default parameter are outside the function body, hence they do not influence the varying property. This is now handled correctly.

  Note: Previously, the MDL core compiler did not always detect invalid MDL code due to these bugs. Hence, old materials might now be (correctly) marked as erroneous. Typically, it is easy to fix them. In most cases the cause is a material or function parameter that is not marked as uniform but is used to compute an argument value of a function inside the body that must be uniform.

42 MDL SDK 2016.1.4, build 261500.12792

42.1 Added and Changed Features

- Extended `IMdl_factory::create_materials()` such that it also supports the function definition for the material constructor as root node of the provided material graph.

42.2 Fixed Bugs

- Fixed `IMdl_factory::create_materials()` such that it now automatically detects if a newly created parameter must be uniform because of the used prototype, and properly fails otherwise.

43 MDL SDK 2016.1.3, build 261500.12088

43.1 Added and Changed Features

- Added the possibility to create instances of `IArray` with length `0`. This can be useful in generic code. Note that it is still not possible to create such attributes.
- The method `IFactory::dump()` supports now MDL annotations, annotation blocks, and type/value/expression/annotation lists when they occur in untyped arrays or structures. Useful for debugging arguments passed to `IMdl_factory::create_variants()` or `IMdl_factory::create_materials()`.

43.2 Fixed Bugs

- Fixed ignoring the gamma value of memory-based textures in `IMdl_factory::create_variants()`.
- Fixed a crash that occurs with zero-length immediate-sized arrays in annotations.
- Fixed a crash if a material or function variant was declared with wrong parameter names. Now an error is reported.
- Added support for memory-based resources in `IMdl_factory::create_materials()`.
- The MDL compiler now generates better error messages if module names in import statements contain syntax errors.

44 MDL SDK 2016.1.2, build 261500.10935

44.1 Added and Changed Features

- Removed `mi_base_assert` from `mi::base::Lock::~Lock()`. The macro might be mapped to an exception and destructors should not throw exceptions.

44.2 Fixed Bugs

- Fixed missing import statement for `tex::gamma_mode` when `IMdl_factory::create_materials()` was used.
- Fixed wrong resource names that were generated when resources were used with `IMdl_factory::create_materials()`.
- Fixed error in MDL compiler when copy constructors of resource types were used (which is legal in MDL). Fixed generation of such extra copy constructors around resource values when `IMdl_factory::create_materials()` was used.
- Fixed result type calculation of the ternary operator inside the MDL compiler that could lead to extra copy constructors.
- Fixed check whether at least one entity of an overloaded set can be imported for a given function. This fixes the problem that `state::rounded_corners()` could not be imported into MDL modules for MDL < 1.3.
- Fixed crash in `IFactory::dump()` for untyped arrays and structs with non-`IData` elements.

45 MDL SDK 2016.1.1, build 261500.9492

45.1 Added and Changed Features

- Support for `state::texture_tangent_(u|v)()` in the GLSL backend has been added.
- Support for `state::geometry_tangent_(u|v)()` in the GLSL backend has been added.
- `state::texture_coordinate()` is now handled like `state_texture_tangent_*()` in the GLSL backend, allowing different behavior depending on backend options.

45.2 Fixed Bugs

- Rare crashes occurring when an MDL user-defined function containing uninitialized arrays was inlined into MDL materials have been fixed.
- A crash in the JIT backend if a great amount of constant data was used inside an MDL material has been fixed.
- A crash in `IMdl_factory::create_materials()` that could occur if a member was selected from a struct typed entity has been fixed.
- Some wrong MDL code generated by `IMdl_factory::create_materials()` in several conditions has been fixed. Some missing imports have been fixed. The generated resource type of parameters of a new material has been fixed.
- Generated names for enum values inside `IMdl_factory::create_materials()` have been fixed. Now the enum value is generated instead of the enum type name (with extra prefixes).
- When a material instance is compiled, the signature of calls inside the arguments is now replaced by their original signature if these are calls to aliases.
- Importing light profiles with default resolution used a lower resolution than necessary for sub types with symmetries. The previous behavior can be restored by explicitly requesting the lower resolution.
- Fixed importing of light profiles of type B without symmetry. The order of values in θ direction was reversed.
- Fixed importing of light profiles of type B with 0/90 degree symmetry. The values for $\phi = 0$ were wrong.

46 MDL SDK 2016.1, build 261500.8353

46.1 Known Bugs

- The `"enable_ro_segment"` option in the LLVM backend is broken in this release. If set to `"on"` (the default), the backend will crash or generate invalid code. As workaround, set it to `"off"`.

46.2 Added and Changed Features

- The `prerelease` parameter of `anno::version()` has now a default value.
- Changed signature of `anno::dependency()` to match that of `anno::version()`.
- Added a new JIT backend option to be able to switch off libdevice linking.
- Added `IMdl_backend::get_device_library()` to retrieve the GPU device library.
- Added support for MDL 1.3 `math::emission_color(color)`.
- Added uniform MDL 1.3 state function `wavelength_min()/wavelength_max()`.
- Removed `limits::WAVELENGTH_MIN/limits::WAVELENGTH_MAX` in MDL 1.3.
- Removed spectral helper functions `base::color_constructor()` and `base::emission_color_constructor()`; their functionality is now provided by the standard library.
- Moved `::df::simple_glossy_bsdf_legacy()` to `::nvidia::df()`.
- Switched off replacement of pre-MDL 1.3 `df::simple_glossy_bsdf()` to `simple_glossy_bsdf_legacy()`.
- Added an option to the JIT code generator to use a pair of `tangent_u`, `tangent_v` arrays instead of one bitangent array in the MDL state.
- Implemented the PTX backend option `output_format`. This allows the PTX backend to generate PTX, LLVM-IR, and LLVM-BC.
- Removal of DB elements has been implemented, see the method `ITransaction::remove()`.
- The class `Type_traits` has been extended such that it also includes the corresponding primitive type for most interfaces in the IData hierarchy. It also provides the reverse mapping.

46.3 Fixed Bugs

- Fixed compilation of texture functions using `texture_3d` type in the JIT backend.
- Fixed export of MDL variants. Ensure that features from a higher MDL version are not used if a module for a lower MDL version is created. The exporter always creates a MDL version that is taken from the original material. Be sure not to use higher language version features.
- When creating a variant through the API, ensure that used entities from default arguments are imported.
- When creating variants, handle predefined types and enums correctly.
- Fixed a crash in the MDL compiler that could occur if an imported function is inlined into a material body.

- Fixed imports of generated materials when created from instances. Fixed wrong imports of built-in types and enum types.
- Fixed a crash in the MDL compiler handling of positional arguments when a variant is declared which is legal MDL.
- Fixed a crash in the MDL compiler related to inlining of function variants.
- The generation of time stamps has been fixed. This affects the behavior of the methods `ITransaction::get_time_stamp()` and `Transaction::has_changes_since_time_stamps()`.
- An explicit constructor from the underlying POD type with different element type has been added to the math classes `Vector`, `Matrix`, and `Bbox`. This makes the POD/non-POD conversion completely transparent on construction (but not on assignment).

47 MDL SDK 2016.1 Beta, build 261500.5273

Initial release.