5.1.0.0

Enhancements

- **Non-Photorealistic Rendering**: a non-photorealistic rendering (NPR) solution is provided as the combination of the `contour` filter and `toon` shader. The `contour` filter draws contour lines using the information provided by the `toon` shader and it works even for reflected or refracted objects. The `toon` shader can be used to obtain a cell animation look. A variety of interesting effects can be achieved by, for example, changing the line width using the `edge_width_scale` parameter, connecting a procedural texture to `mask_color`, or using stylized highlights. (#5591, #6848)
- **Adaptive sampling**: Arnold now has the capability of adapting the sampling rate of each pixel when the `enable_adaptive_sampling` render option is enabled, allowing it to dedicate a greater number of camera samples (and thus also a greater amount of render time) to the pixels that show a greater variation in their sample values. When used, all pixels will receive a sampling rate of at least `AA_samples`, but no more than `AA_samples_max`. The adaptive sampler's sensitivity to noise may be controlled through the `AA_adaptive_threshold` render option, where lower threshold values will apply higher sampling rates to a greater number of pixels. The `-asmax` command-line option has been added to `kick` as well. (#3165, #6090, #6686, #6883)
- **Denoiser (noice)**: a stand-alone, post-process denoiser executable called `noice` is now bundled with Arnold. This is a high-quality denoiser that takes into account multiple frames and multiple light AOVs. It requires variance information for all AOVs and optionally uses normal, depth and albedo. (#6322)
- **Denoiser (OptiX)**: the fast, GPU-powered Nvidia OptiX AI denoiser is now available in Arnold as a filter called `denoise_optix_filter` (#6513). Note that it is not available on OSX. This filter has one parameter `blend` which is a float and controls the linear interpolation between the denoised and original pixel values. The filter can be applied to an output just like other filters, with the following limitations:
  - The filter must be unique for each output it is applied to. Meaning that you must create a new OptiX denoise filter for each output you wish it to be applied to.
  - The filter will only work with single-channel EXR images.
  - Outputs with the OptiX denoise filter applied will be received by the driver later than the outputs without. As the OptiX denoiser works on full frame images, `driver_prepare_bucket` and `driver_process_bucket` will be called again after the full frame has completed providing bucket data for only the denoised output.
  - If you want to render the same AOV with and without the OptiX denoise filter, we have added a feature to allow you to do so and avoid AOV name conflicts. You can add the suffix `_denoise` onto the end of an AOV name and the filter will source the name from the original AOV. For example if we wanted to apply the OptiX denoise filter to the RGBA AOV, but already had an RGBA AOV added to the outputs, we could add an AOV named `RGBA_denoise` and the OptiX denoise filter will source the RGBA AOV as the input.
- **Alembic procedural**: a procedural node called `alembic` which is capable of reading Alembic .abc files has been added. It resides in an external dynamic library that is by default located in the "plugins" directory of the Arnold core package. (#6547)
- **Automatic loading of plugins from Arnold installation**: Arnold looks for a directory "../plugins" relative to where the core shared library is installed and automatically loads procedurals, shaders, etc from that location. This is also the default location of the newly introduced Alembic procedural. (#6808)
- **Operators**: Operators are a new node type which perform per-object (node) parameter assignments and overrides, including late-bindings and deferred overrides on procedurally generated nodes. Operators can also create nodes and carry out general scene inspection and modifications. Operators can be chained together in a graph which is evaluated from a given target operator, set using `AiOpSetTarget (node)` or `kick -op node_name`. Multiple disconnected operator graphs can exist in the scene, where only the graph connected to the target operator will be evaluated for rendering. Operators can be ignored by setting `options.ignore_operators` to true or using `kick -iops`. Some operators provide a selection parameter which determines, using a wildcard expression, what nodes are processed by the operator. A series of operator nodes are now available: `materialx`, `set_parameter`, `disable`, `collection`, `switch_operator`, and `set_transform`. (#6209, #6210, #6530, #6606, #6662, #6699, #6676, #6700, #6701, #6878)
- **More efficient texture mapping**: the performance of certain types of texture lookups, including opacity masks, skydome_lights, and certain kinds of specular and diffuse rays has been optimized. (#6387, #6391, #6480, #6603)
- **Faster scene initialization**: scene initialization and update can be much faster as they are now performed in parallel by default. Multiple threading issues were fixed that also make the scene initialization code more stable. Finally, procedural contents are initialized asynchronously, in parallel with the rest of the nodes, to make full use of all available threads. In the case of interference with non-threadsafe scene construction techniques involving custom procedurals, parallel initialization can be manually disabled by setting `options.parallel_node_init` to false. (#5724, #5406, #6744)
- **Difference filter**: an auxiliary `diff_filter` for denoising that measures AOV variance has been added. (#6462)
- **EXR metadata for AOVs**: additional metadata tags (filter, filter width, LPE, original AOV source) have been added to describe output AOVs. (#6461)
- **Matte shader opacity**: the `matte` shader has gained an opacity parameter, so it can honor transparency the same way the built-in `matte` object parameter does. (#6415)
- **Per-lightgroup shadow mattes**: The `shadow_matte` shader has a new `aov_group` parameter that when enabled makes the shader sensitive only to lights with a matching `aov` setting. (#6609)
- **Trace sets in ambient occlusion shader**: the `ambient_occlusion` shader now supports the `trace_set` parameter to specify which objects are included or excluded. (#6602)
- **`extra_samples` in hair shader**: the `standard_hair` shader now supports the `extra_samples` parameter to use additional GI samples on a per-shader basis. (#6443)
- **Improved uniformity in hair shader**: like the classic `hair` shader, the `standard_hair` shader will now display a uniform result over a strand's cross-section, which can reduce sampling noise. (#6444)
- **`image` tile tags optional offset**: the `<tile>` tag in the `image` shader's `filename` now accepts an optional tile offset, instead of being hard-coded to one, e.g. `<tile:0>` is now possible. (#6429)
- **New wrap mode in `image` shader**: similar to the preexisting `black` wrap mode, there is now a `missing` wrap mode where lookups to the `image` shader that are outside the texture will use the `missing_texture_color`. (#6430)
- **ID AOVs in `standard_surface` and `standard_hair`**: the `standard_surface` and `standard_hair` shaders now support ID AOVs. These are useful for creating mattes. (#6693)

- **Face mode option in `color_jitter`**: with the new `face_mode` option, color can be randomized per quadrangle as well as triangle. (#6495)
- **Reference positions from `rgb`/`rgba` array**: procedural texture shaders (`noise`, `flakes` `triplanar`, and `car_paint`) can now read reference positions (`Pref`) from an `rgb` or `rgba` array as well as a `vector` array. (#6729)
- **Arbitrary name for reference positions**: users now can specify the name of the reference position user-data array in procedural texture shaders such as `noise`. Previously, the name was hard-coded as "Pref", which is still the default. (#6709).
- **New AOV-write shader**: an `aov_write_rgba` shader has been added which complements the existing `rgb`, `float` and `int` variations of this shader. (#6639)
- **Layer shaders**: the newly added `layer_float` and `layer_shader` shaders can be used to mix float values and closures respectively. `layer_rgba` allows to composite textures. The maximum number of layers in these shaders is limited to 8. (#6549)
- **`normal` parameter in `passthrough` shader**: the `passthrough` shader now has a `normal` parameter that allows for the assignment of a normal or bump map that affects the entire network of shaders it is connected to. (#6435)
- **Self-intersection detection in OSL trace**: OSL shaders can now more easily inspect the self-intersection status of probe rays via the "hitself" trace message like so: `getmessage("trace", "hitself", hit);` This function assigns a 1 or 0 to the `hit` variable depending on whether the ray intersects the same object or not. (#6326)
- **Oren-Nayar transmission in OSL**: the `translucent` closure in OSL, which is based on a back-facing Oren-Nayar shading effect, can now be fed a second parameter indicating the surface roughness in a similar fashion to the `oren_nayar` closure. (#6424)
- **`ginstance` can override `step size`**: `ginstance` nodes can now override the `step_size` parameter of volume containers such as points, polymeshes, cubes and spheres. This allows for example an instance of a volume shape to be shaded as a surface (by setting the instance step size to zero, opaque off, and using a surface shader). (#6627)
- **Less confusing stack traces**: Arnold would often misleadingly print out `AiShaderGlobalsEdgeLength` or `AiCreateAtStringData_private` in the call stack. These functions should no longer be displayed. (#6439)
- **Structured statistics**: render statistics can now be output to JSON files at the end of each render pass, either appended to or overwriting an existing .json file. This is much easier for tools to inspect rather than attempt to parse out the raw-text statistics in the logs that were meant for human consumption. (#6108)
- **Better time statistics**: additional timing statistics, organized by both nodes and categories, will now be output. This makes it possible to know which objects are most expensive to render and what parts of the renderer took the most amount of time. Detailed information about rendering performance can be output to a file in JSON format, such as "my_profile.json", by calling `AiProfileSetFileName("my_profile.json")` or `kick -profile "my_profile.json"`, and then visualized in Google's Chrome web browser "chrome://tracing/". (#5106)
- **Frame and fps global options**: current frame and frames per second attributes have been added to options as `options.frame` and `options.fps`. This information will also be exported to rendered images as EXR metadata. (#6474)
- **`lmutil`**: a licensing tool called `lmutil`, which can be used to diagnose and solve certain FlexNet/AdLM/Clic licensing issues, is now distributed in the Arnold core package. (#6528)
- **Progressive refinement (EXPERIMENTAL)**: a new rendering mode that completes a render call in multiple passes has been added to Arnold. During each of the intermediate passes, drivers that do not output to a file will be invoked after each tile has completed, which allows for display drivers to show a result whose noise progressively converges towards the result at the final AA sample settings. This mode can be enabled through the `enable_progressive_render` render option (default: `false`). Note that, in its current unoptimized state, this mode is not recommended for batch rendering at high AA samples, as the final passes can take very long to filter. This will be addressed in a future update. (#6146)
- **Node dependency graph (EXPERIMENTAL)**: a new node dependency graph allows tracking references between nodes, so that when a node is deleted or replaced, it can be automatically changed anywhere the node may have been referenced in the scene. It can be enabled using `options.enable_dependency_graph`. This is still a work in progress, so there are some unsupported cases, which will be addressed in a future release. (#6437)

API additions

- **Shader compatibility**: even though this is technically an API-breaking release, Arnold 5.0 shaders are still compatible with 5.1 and can continue to be used without being recompiled. (#6475)
- **Render session type**: `AiBegin()` has gained an argument specifying whether the session is for offline, batch rendering (`AI_SESSION_BATCH`) or for interactive/IPR rendering (`AI_SESSION_INTERACTIVE`). Arnold will perform certain optimizations during batch rendering because it expects nodes will not be modified after rendering, but which may leave the scene missing information. Interactive rendering will leave all information intact so that nodes may be edited as needed after rendering and re-rendered again. (#6234)
- **New render control API**: Invoking a render now happens asynchronously through `AiRenderBegin()`. This takes an optional render update callback which will be called before and after each render pass and in which the scene may be modified for interactive rendering. During interactive rendering, only the main output will have pixels sent to, until the final pass where all outputs are activated. Rendering paused via `AiRenderInterrupt()` can be restarted with `AiRenderRestart()` or restarted in the callback. And finally, when rendering is done or needs to wrap up, `AiRenderEnd()` must be called. The current render status is obtained with `AiRenderGetStatus()`. Please see the API documentation for more information on the new render control API. Note: the old `AiRender()` and `AiRendering()` API functions are now deprecated. (#6234, #6226, #6542)
- **Structured statistics API**: two functions have been added to set up output of render statistics to .json files, `AiStatsSetFileName(const char*)` and `AiStatsSetMode(AtStatsMode)` to control JSON stats output. Two additional functions to get the filename and mode are also available. The two available modes are to output JSON and overwrite any existing data in the stats file, or to append additional render pass statistics and keep any previous stats in the file. (#6108)
- **Better time statistics API**: user functions can also be profiled by calling `AiProfileBlock ("my label", my_AtNode);` at the start of a code block that needs to be timed. This can be called with just a string literal, just an `AtNode`, or both. This profiler has low overhead, so is generally safe to use without incurring any noticeable slowdown. (#5106)
- **Device selection API**: New APIs for selecting devices Arnold may use during the rendering process. Currently functionality for these APIs is only for selecting GPUs to use for denoising. See `ai_device` for full details (#6329, #6539, #6580, #6586, #6623, #6685, #6767)
    - `AiDeviceSelect` - Explicitly select which devices Arnold may use during the rendering process.
    - `AiDeviceAutoSelect` - Auto selected devices that can be used via the options `default_gpu_names` and `default_gpu_min_memory_MB`
    - `AiDeviceGetSelectedType` - Queries the currently selected render device type. Currently only `AI_DEVICE_TYPE_CPU`
    - `AiDeviceGetSelectedIds` - Queries the currently selected device IDs

    - `AiDeviceGetCount` - Queries the number of devices by type (`AI_DEVICE_TYPE_CPU`/`AI_DEVICE_TYPE_GPU`)
    - `AiDeviceGetIds` - Queries valid device IDs by type (`AI_DEVICE_TYPE_CPU`/`AI_DEVICE_TYPE_GPU`)
    - `AiDeviceGetName` - Queries device name
    - `AiDeviceGetMemoryMB` - Queries device memory information. (Total/Free/Used)
- **AiNodeReplace**: the new `AiNodeReplace` API function allows in-place substitution of a node, updating references from other nodes based on the new (experimental) dependency graph. So, for example, a shader can be replaced by another, automatically updating all nodes using the first shader. There are still some limitations. For example, instanced nodes are not yet supported. (#6466)
- **OSL support for writing to AOVs**: OSL shaders may now write out to AOVs via `debug()` closures. (#5466) The syntax looks like this:

```
shader surface_shader_osl(output closure color result = 0)
{
   // The weight of the debug closure will be added to the DBG1 AOV
   result = cellnoise(u * 10, v * 10) * diffuse(N) + color(0, 1, 1) * debug("DBG1");
}
```

Incompatible changes

- **Removed `P` parameter in `triplanar`**: the `P` parameter was broken and has therefore been removed. This parameter was provided to read reference positions exported as a color array by connecting `user_data_rgb` and `rgb_to_vector`. This can now be done directly without these nodes. (#6264, #6709 #6729)

Bug fixes

- #6264 reference positions do not work in triplanar
- #6302 Clear up indirect node usage
- #6396 small bug fixes in car_paint
- #6402 zero-radius problem in random-walk SSS
- #6576 nurbs and polymesh common parameters not copied
- #6730 Fix bug in procedural name scope that was causing a race condition with parallel_node_init
- #6731 Fix race condition in overrides
- #6732 Fix bug in transformation of lights contained in procedurals when using parallel_node_init
- #6734 Normal map issue using strength parameter with back facing normals
- #6736 Artifact with intersecting object in polymesh-based volumes
- #6741 Fix race conditions in node name manipulation code
- #6752 tag with index but without default value crashes
- #6756 implement missing `bsdf_merge` for hair BCSDFs
- #6766 Python AtArray get and set functions crashing on error due to missing arguments
- #6777 AiNodeLookUpByName and AiNode crash with non-procedural parent
- #6799 Point and Vector functions in ai_matrix.py not returning correct type
- #6810 NaN in Zinke BCSDF
- #6837 high res opacity masks sometimes have incorrect regions
- #6872 kick -ar (aspect ratio) is broken
- #6652 AiMakeTx() fails with single channel input and DWAA compression
- #6738 Fix random crash with unnamed nodes#6791kick -ipr default value not working