3.2.0.0

Milestone 3.2

This release has important changes in the way we build the Arnold binaries:

- **C++ compiler required:** For years, Arnold has had a strictly C-only API. The API is still mostly C, but we now have a few subtle uses of C++ features, most notably arithmetic operators for colors and vectors. This means that applications that link to Arnold must be built with a C++ compiler/linker. If your application was using .c files only, you may have to rename them to .cpp.
- **Linux:** We have switched our primary development platform from FC10 to CentOS 5.4. This addresses the long-standing issue of glibc /stdlibc++ compatibility with bigger studios that for legacy reasons are stuck with older versions of Linux.
- **Windows:** Switched development environment from VS2005 to VS2008. This means that Arnold now links with the VS2008 runtime libs. The compiler itself is still the same (icc).

Enhancements

- **Reduced memory usage in instances:** Improved the storage of user-data to consume less memory, especially on instances. The savings is greatest when the instances inherit user-data defined on the source geometry. (trac#1755)
- **Vector-displacement support for autobump:** The autobump feature was previously limited to displacement along the normal direction. It now works properly when displacing along an arbitrary vector direction. (trac#1760)
- **Faster adaptive subdivision:** Optimized adaptive subdivision for regular patches. This results in a 1.6x speedup in subdiv time. Note that there is no speedup for regular, non-adaptive subdiv, i.e. pixel_error=0. (trac#1757)
- **Subdiv limit normals:** Subdivision surfaces now use limit normals instead of face-averaged normals. This is specially important with adaptive subdivision, reducing popping in animation. (trac#1032)
- **Per-face subdivision level:** It is now possible to specify subdivision iteration levels on a per-face basis by using the built-in subdiv_face_iterations uniform user-data, which is set as an array of AI_TYPE_BYTE values. (trac#1676)
- **Reflection lines visualization:** A new reflectline color mode has been added to the utility shader which shows reflection lines that can be used to judge the smoothness of a surface. (trac#1759)
- **Added render option abort_on_license_fail:** It is now possible to tell Arnold to error out when a valid license is not found, instead of rendering a watermarked image. (trac#1736)
- **Interactively changing the number of threads:** Arnold now allows users to interactively change the thread count between calls to AiRender(). (trac#933)
- **Increased thread limit:** The hardcoded limit of 16 maximum threads has been bumped up to 64 threads. (trac#1771)
- **Faster .ass writing in big scenes:** Writing out an .ass file for a scene with many nodes was taking quadratic time due to several uses of AiNodeLookUpByPointer(), which performs a linear search over all nodes in the scene. This was found in a scene with 800k nodes. (trac#1754)
- **Optional tiled EXR output:** A new parameter, tiled, has been added to the driver_exr node. This gives users the ability to generate untiled, scanline-based EXR files. The default value is set to TRUE for backwards compatibility. (trac#1777)
- **EXR compression now defaults to zip:** The default compression type in the driver_exr node has been changed from none to zip. (trac#1764)
- **Upgraded OIIO to 0.7:** Among other fixes, this includes support for reading the *Softimage PIC* format, as well as a fix for a Linux crash when reading many TIFF texture maps in a multi-threaded environment. (trac#1767, trac#1781)

API additions

- **Arithmetic operators for vector and color types:** Now that we require clients to use a C++ compiler, we can take advantage of operator overloading. For starters, we have added support for the most common arithmetic operators in the AtColor and AtVector types. It is finally possible to write succint code like AtColor a = b + c * -d. This would previously require multiple lines of C macros like AiColorAdd(), AiColorScale() etc. This can typically reduce the size of shader source code by 15% while improving readability, maintenability and, in some cases, resulting in better performance (such as in cases where the equivalent C macro would result in redundant evalutions of a complex subexpression in one of the macro arguments). (trac#1723, trac#1734)
- **AI_TYPE_NODE:** It is now possible to create node parameters that unambiguously point to an AtNode object. This is in contrast to the existing AI_TYPE_POINTER, which can point to an arbitrary location in memory (e.g. a raw memory buffer or a function pointer). This distinction allows for stronger type checking and also allows various internal optimizations (like faster .ass file writing). The new type is for all practical purposes compatible with the old AI_TYPE_POINTER type, and in fact shares the same accessors: you must use AiNodeSetPtr() and AiNodeGetPtr() to respectively set and get a parameter. A number of existing node parameters such as polymesh.shader, options. background etc have been changed from AI_TYPE_POINTER to AI_TYPE_NODE. Use AiParameterNODE() to declare node parameters of this new type. (trac#1769)
- **AiNodeClone():** This new API lets you "clone" or copy an existing AtNode. (trac#1775)
- **AiNodeEntryIterator():** Provided a new API for iterating over node entries, in the same way as we already have for nodes. (trac#1758) Same as with the node iterator, you can request only a subset based on a bitmask of node types. For example, to print all installed cameras and shaders along with how many nodes have been created of each:

```
AtNodeEntryIterator *iter = AiUniverseGetNodeEntryIterator(AI_NODE_CAMERA | AI_NODE_SHADER);
while (!AiNodeEntryIteratorFinished(iter))
{
   AtNodeEntry *nentry = AiNodeEntryIteratorGetNext(iter);
   printf("\n%s (%d)", AiNodeEntryGetName(nentry), AiNodeEntryGetCount(nentry));
}
AiNodeEntryIteratorDestroy(iter);
```

- **AiV3IsSmall():** This macro has been added for consistency with the existing AiColorIsSmall(). (trac#1773)
- **New Python bindings:** Implemented vector and matrix methods in Python bindings. (trac#1766)

Incompatible changes

- Needs a C++ compiler!
- Removed AiNodeLookUpByPointer(). This function performs a linear search over all nodes in the scene, which is prohibitively expensive in big scenes. Use AiNodeGetName() instead. (trac#1770)
- Removed AiM4Print(). (trac#1774)
- Deprecated AiGetNumInstalledNodes() and AiNodeEntryLookUpByIndex(). You should use AiUniverseGetNodeEntryIterator() instead. (trac#1758)
- Removed unsupported shaders blend and ramp. (trac#1747)
- Removed raydump and (unused) adaptive antialiasing options: (trac#1742, trac#1744)
```
pow2_sampling
AA_samples_max
AA_criterion
AA_threshold
AA_filter
AA_filter_width
raydump_filename
raydump_x
raydump_y
```

- Renamed several node parameters. The old names still work, but you'll get a deprecated warning message: (trac#1746)
```
image.image_map      --> filename
object.inv_normals   --> invert_normals
ignore_tmaps         --> ignore_textures
ignore_atm_shaders   --> ignore_atmosphere
message_num          --> max_shader_messages
shadow_term_fix      --> shadow_terminator_fix
fixture_threshold    --> luminaire_bias
TM_tgamma            --> texture_gamma
TM_lgamma            --> light_gamma
TM_sgamma            --> shader_gamma
GI_hemi_samples      --> GI_diffuse_samples
GI_specular_samples --> GI_glossy_samples
```

- Renamed the -hs kick option to -ds. This follows the renaming of GI_hemi_samples to GI_diffuse_samples. (trac#1746)
- Renamed the -ss kick option to -gs. This follows the renaming of GI_specular_samples to GI_glossy_samples. (trac#1746)
- The visible boolean parameter in the sky node, which only worked for camera rays, has been renamed to visibility and changed to a generic ray visibility bitmask, exactly like the visibility parameter in the geometry nodes. You can now make the sky visible to any combination of ray types. Old scenes that were disabling the visible flag will render incorrectly, i.e. the flag will be ignored and the sky will be visible to camera rays. You can fix old scenes like this: (trac#1780)
```
- visible off
+ visibility 65534    # AI_RAY_ALL - AI_RAY_CAMERA = 65534
```

- The dither_amplitude global option has been moved to each of the output driver nodes, e.g. driver_tiff.dither_amplitude. (trac#1784)
- The AiQuantize*() API now takes an additional parameter dither_amplitude. The new signature is: (trac#1782)
```
AI_API AtByte AiQuantize8bit(AtInt x, AtInt y, AtInt i, AtFloat value, AtFloat dither_amplitude);
```

Bug fixes

#1781	Crash in OIIO related to multithreaded texture access
#1762	Crash when passing a file name with no extension to an output driver
#1752	missing points primitive under 'geometric elements' stats
#1786	crash when changing the AA filter with kick -af
#1785	crash in AiEnumGetValue() and AiEnumGetString() with NULL enum
#1768	Fix alpha compositing for auto_transparency mode
#1765	Transparent objects are showing up on the depth AOV
#1750	displacement in world coordinates produces autobump artifacts
#1748	crash with autobump and ignore_smoothing
#1745	change default texture_max_open_files to 100
#1741	artifacts in curves with duplicated b-spline end points
#1740	bump2d doesn't work with negative height values
#1737	crash in standard shader with negative Phong exponent
#1725	AiMsgUtilGetUsedMemory() should return a 64-bit int
#1722	AiMsgUtil* functions are not exported properly
#1720	bump3d shader is inverted