4.2.12.0

Milestone 4.2.12.0

Enhancements

- **Memory savings for vertex normals**: The storage of polymesh vertex normals has been optimized, reducing memory use by 50% in typical production scenes. (#4987, #4992)
- **Improved Russian Roulette**: The standard shader now uses more aggressive Russian roulette termination. This reduces render time but increases noise, so to get renders with similar noise levels as before either AA_samples orGI_diffuse_samples, GI_glossy_samples and GI_refraction_samples can be increased. For interior scenes with high GI depth, this is a big win and we have found 2x to 3x faster renders for similar noise levels. (#2445, #4944, #4945)
- **Empirical SSS profile**: A more physically accurate subsurface scattering profile is now available. With a single layer, it can capture both surface detail and deep scattering. In the standard shader it can be used by setting the new parametersss_profile to empirical. A new AiBSSRDFEmpirical API function has been made available to shader writers. (#4229)
- **GGX Microfacet**: A new specular_distribution was added to the standard shader, with options to use the existing Beckmann distribution, and a new GGX distribution. The GGX distribution has a sharp highlight with a long soft tail, which better matches many real world materials. Using GGX also enables more accurate per-microfacet Fresnel. New microfacet BSDF functions are available in the shader API, see below. (#4950)
- **Per-light AOVs for volumetrics**: This feature allows modifications to the intensity and color of different lights in compositing, without having to re-render. The new aov string parameter on lights writes out the light contribution to a separate AOV with a corresponding name. For example, all lights with parameter aov "fill" will contribute to the RGBA_fill and volume_fill AOVs, which then contain a subset of the light from the RGBA and volume AOVs respectively. Emission from other sources and lights that have not been assigned an aov name will contribute to the RGBA_default and volume_default AOVs. All light AOVs can be output to a single EXR driver using output syntax like RGBA_* RGBA filter driver. For the time being, light AOVs are only supported for volumes, and lighting from surfaces and the atmosphere will end up in the default light AOV. A maximum of 8 different light AOVs are supported, although a given AOV can contain a bundle of any number of lights. (#4935, #4971)
- **<tile> and <attr> tags in image shader**: The built-in image shader node now supports the <tile> tag, and will do Mudbox-style tag replacement of the tag with _uC_vR where C and R are the column and row of the tile, respectively. The attribute tag is also supported in the form <attr:name index:name default:value>. The tag will look for the named user data (as a string). The index and default tokens are optional; if the index is used, the UINT user data is found first, and the main attribute then must be an array of strings it indexes into. Among other techniques, this allows e.g. facesets, where you can list the faceset names once each in a constant array, and then have a uniform UINT assigned to each face specifying which faceset the face is a member of. Finally, the default, if present, is substituted if the user data cannot be found for any reason. (#4787)
- **Multiple tags in image shader**: The image shader node now also supports replacing up to five dynamic tags. Combinations of multiple attribute, tile or UDIM tags are allowed. It generally makes sense to only have one tile or UDIM tag, but if there are multiple tile/UDIM tags the last one generates the final UV coordinates into the texture. (#4270)
- **Faster light samplers**: Light sampling code has been reorganized internally for increased consistency and cleanliness, resulting in slight performance improvements to light sample generation. Certain types of lights and lighting setups, such as static IES/photometric lights, large-area rim lights, or skydome lights with a constant color, may see even greater gains; we have seen up to 15% faster render times. (#2663, #4965, #4967, #4972, #4988)
- **Adaptive subdiv for non-linear cameras**: Adaptive subdivision is now supported for all built-in cameras, even non-perspective ones (cylindrical, spherical, etc). Adaptive subdiv is not yet supported for non-perspective *custom* cameras, but a reasonable workaround is to use subdiv_dicing_camera with the built-in camera that better matches the custom camera. (#4528)
- **Vector/point to point2 parameter conversion**: Shaders with AI_TYPE_VECTOR or AI_TYPE_POINT output types will now convert by dropping their *Z* component when linked to AI_TYPE_POINT2 parameters instead of doing nothing. (#5001)
- **Negative thread counts**: Negative numbers in options.threads are now allowed. If specifying 0 threads means use all cores on a machine, then negative numbers can mean use all but that many cores. For example, threads=-2 means use all but 2 cores, while threads=2 means only use 2 cores. This is useful when you want to leave one or two cores for other tasks. One example of this is so that Maya can be more responsive while Arnold is rendering in the Render View. (#5012)
- **Better movement scale in kick -ipr**: When using kick -ipr q for Quake-style IPR navigation, the default movement scale will be picked based on the scene size, so it will be easier to use the WASD keys to move forward/backward and side to side. Note that this option used to be called "-interactive". (#4974)
- **Render bounds in EXR metadata**: Non-zero image extents are now available in EXR metadata in all configurations (deep/flat, tiled /scanline). The smaller processing area will speed up 2D compositing operations accordingly. These bounds can be queried in Nuke to set up a Crop or DeepCrop node using the expressions below. (#4979)

```
crop.box.x : [metadata exr/arnold/bounds_min_x]
crop.box.y : height - [metadata exr/arnold/bounds_max_y]
crop.box.r : [metadata exr/arnold/bounds_max_x] + 1
crop.box.t : height - [metadata exr/arnold/bounds_min_y] + 1
```

- **Upgraded OIIO to 1.5.22**: Aside from several minor bugfixes, the OIIO maketx utility in "update" (-u) mode now remakes the file even with matching dates, if different arguments or maketx versions are used. For instance, if maketx foo.jpg -u -d uint8 was used to create an 8bit texture and then maketx foo.jpg -u -d float is used, the previous maketx would not have generated the float texture since the underlying foo.jpg was never modified. (#4999, #5025)

API additions

- **Russian Roulette**: Custom shaders can now be optimized to take advantage of Russian roulette termination. AiTrace, AiIndirectDiffuse, AiBRDFIntegrate, AiOrenNayarIntegrate and AiMicrofacetBTDFIntegrate now take a color weight parameter, which will be multiplied with the resulting color and used to track the path throughput for Russian roulette termination. For best results this weight should include all factors that would previously be multiplied with the resulting color, like the component color or Fresnel weights. (#4944)

```
AtColor indirect = fresnel_weight * diffuse_color * AiIndirectDiffuse(&sg->Nf, sg); // old
AtColor indirect = AiIndirectDiffuse(&sg->Nf, sg, fresnel_weight * diffuse_color); // new
```

- **Empirical BSSRDF**: The API function works similar to the existing Cubic and Gaussian BSSRDFs, using raytraced SSS to render an arbitrary number of weighted profiles. The parameters are different however. The mean free path is used instead of a radius, controlling the average distance light scatters rather than the maximum distance. Further, the surface albedo is specified separately from the profile weight. albedo influences the shape of the profile, and as such should only describe properties of the volume below the surface, while weight can be used for Fresnel or other layering weights.

  ```
  void AiBSSRDFEmpirical(const AtShaderGlobals* sg, AtRGB& direct, AtRGB& indirect, const float* mfp,
  const float* albedo, const AtRGB* weight, unsigned int num = 1);
  ```

  Example usage, using a separate profile for each color channel:

```
float sss_fresnel = 1 - AiFresnelWeight(...);
AtColor sss_mfp = AiShaderEvalParamRGB(p_sss_mfp);
AtColor sss_albedo = AiShaderEvalParamRGB(p_sss_color);
AtColor sss_weight[3];
sss_weight[0] = AiColor(sss_fresnel,           0,           0);
sss_weight[1] = AiColor(          0, sss_fresnel,           0);
sss_weight[2] = AiColor(          0,           0, sss_fresnel);
AtColor sss_direct, sss_indirect;
AiBSSRDFEmpirical(sg, sss_direct, sss_indirect, &sss_mfp.r, &sss_albedo.r, sss_weight, 3);
```

- **Microfacet BSDFs**: New microfacet BRDF API functions are available with support for both Beckmann and GGX distributions and per-microfacet Fresnel. (#4950)

```
AtColor  AiMicrofacetMISBRDF(const void* brdf_data, const AtVector* indir);
float    AiMicrofacetMISPDF(const void* brdf_data, const AtVector* indir);
AtVector AiMicrofacetMISSample(const void* brdf_data, float randx, float randy);
void*    AiMicrofacetMISCreateData(const AtShaderGlobals* sg, int distribution, const AtVector* u,
float eta, float rx, float ry);
```

  These mostly work the same as the existing Cook-Torrance BRDF. The new distribution parameter must be set to AI_MICROFACET_BECKMANN or AI_MICROFACET_GGX, while the unused v tangent for anisotropy was removed. The eta parameter is used to include per-microfacet Fresnel in the BRDF, setting it to zero disables Fresnel. When using per-microfacet Fresnel in existing shaders, be sure to remove any other Fresnel factors to avoid applying Fresnel twice to the same BRDF. Additionally there is an API function to retrieve the average Fresnel reflectance over the microfacets, which can be used for weighting for example a diffuse component below the specular layer.

```
AtColor  AiMicrofacetMISAverageFresnel(const AtShaderGlobals* sg, const void* brdf_data);
```

  An updated BTDF integration function is also available, with similar changes as the BRDF functions. The use of indices of refraction has been changed, instead requiring a relative index of refraction and a boolean to specify if we are entering or exiting the object. Per-microfacet Fresnel can be included by setting fresnel to true. Non-zero values for dispersion enable chromatic dispersion.

```
AtColor AiMicrofacetBTDFIntegrate(const AtVector* N, AtShaderGlobals* sg, int distribution, const
AtVector* u, float rx, float ry, float eta, bool entering, float dispersion, bool fresnel, AtColor
transmittance, const AtColor& weight);
```

  Conversion from the previous index of refraction parameters to the new ones is as follows:

```
float eta = (eta_i == eta_o || eta_o == 0.0f) ? 1.0f : eta_i / eta_o;
bool entering = (AiV3Dot(sg->N, sg->Rd) < 0);
```

Incompatible changes

- **Don't discard shadowed samples from null-area lights**: In order to make the overall behavior of lights more consistent and to facilitate their use in shadow mattes, samples coming from lights that have no area (like distant, point and spot lights with radius 0) will no longer be discarded from light sampling loops when they are in shadow. (#2663, #4825)
- **Point lights with radius > 0 now single-sided**: To make their behavior more consistent with Arnold's other area lights, point lights with non-zero radius (i.e. spherical lights) are now single-sided and will no longer illuminate their interior. (#4964)
- **Shadow terminator fix**: The technique used to mitigate shadow terminator artifacts in low-poly objects has been improved when dealing with concave, transparent or translucent surfaces. The obscure global option shadow_terminator_fix has been removed, since it no longer makes a difference in those cases. (#2781, #4981, #5010)

Bug fixes

Ticket	Summary
#2663	lights' usage of sg->Li, sg->Liu and sg->Lo is inconsistent
#4929	crash when aborting while subdivision
#2781	Improved shadow terminator fix for transparent or concave surfaces
#4270	Support multiple token replacements in texture path
#4948	Bump shaders overwrite alpha of RGBA shader
#4949	Crash when evaluating a link to an array element after being unlinked
#4951	Crash tracing camera rays from shaders
#4952	Russian roulette skips standard shader indirect light in rare cases
#4953	Bad width for wireframe seen through refraction or reflection
#4954	Render metadata should have short names for maximum OpenEXR compatibility
#4956	Crash when using shaders that output arrays
#4958	Roughness clamping incorrect with indirect glossy bounces
#4959	Print better info when getting killed by external process, but not crashing
#4977	scene bounds for plane, cylinder, and disk objects incorrect
#4989	Kick crash when outputs are disabled
#4997	Bounding box given by options.curved_motionblur is clipped
#5002	crash in the image shader if there are more than 100 UDIM rows
#5020	crash in BVH if it contains a primitive with invalid bounds
#5024	closest filter should support pointer/node type
#4964	make point lights with radius > 0 single-sided
#4978	Kick window unexpectedly closes on OS X 10.11
#5007	out of range vertex index causes crash