

1. Arnoldpedia .....	4
1.1 Introduction .....	5
1.2 Technical Specifications .....	6
1.3 Arnold 5 Porting Guide .....	8
1.4 API Examples .....	9
1.4.1 Creating and Rendering a Scene .....	10
1.4.2 Custom Gaussian Filter .....	14
1.4.3 Iterate an Arnold Array With the Python Bindings .....	17
1.4.4 Updating and Re-Rendering With Python .....	18
1.4.5 Using the PointCloud API .....	21
1.4.6 Writing a Camera Node .....	25
1.4.7 Writing a Driver .....	28
1.4.8 Writing a Light Filter .....	32
1.4.9 Writing a Raw Driver .....	36
1.4.10 Procedurals .....	40
1.4.10.1 Implementing a Random Pointcloud Procedural .....	41
1.4.11 Shaders (API) .....	47
1.4.11.1 Arnold Metadata Files .....	48
1.4.11.2 Create a DLL that Contains Multiple Shaders .....	49
1.4.11.3 Create an Empty VS 2008 Project to Compile a Shader into a DLL on Windows .....	51
1.4.11.4 Create an Empty XCode 4 Project to Compile a Shader into a Dynamic Library on Mac OSX .....	55
1.4.11.5 Creating a Shader Project with Xcode on Mac OSX .....	68
1.4.11.6 Creating Refracted Rays With Proper Ray Derivatives .....	79
1.4.11.7 Implementing MIS-sampled BRDFs .....	80
1.4.11.8 Shaders Library .....	82
1.4.11.9 Understanding GI With a Basic Shader .....	83
1.4.11.10 Using Trace Sets .....	89
1.4.11.11 Writing A Vector Displacement Shader .....	93
1.4.11.12 Writing Your First Shader .....	103
1.5 API FAQ .....	105
1.6 Arnold Scene Source (ass) .....	110
1.6.1 .ass File Examples .....	115
1.7 Command Line Rendering (kick) .....	117
1.8 Displacement .....	122
1.9 Plugins .....	126
1.9.1 Creating a Simple Plugin .....	127
1.9.2 Multiple Nodes in a Library .....	132
1.9.3 Metadata Files .....	135
1.9.4 Camera Nodes .....	136
1.9.5 Procedural Nodes .....	142
1.9.5.1 Random Flake Procedural .....	143
1.9.5.2 Implementing a simple instancer procedural .....	149
1.9.5.3 Large Datasets from Procedurals .....	153
1.9.6 Filter Nodes .....	162
1.9.7 Driver Nodes .....	165
1.9.7.1 Simple Driver .....	166
1.9.7.2 Display Driver .....	170
1.9.7.3 Raw Driver .....	175
1.10 Rendering .....	179
1.10.1 How to Read a Render Log .....	180
1.10.2 Farm Rendering .....	189
1.10.3 Removing Noise .....	190
1.10.4 Removing Noise Workflow .....	200
1.10.5 What is Sampling? .....	204
1.11 Shaders .....	212
1.11.1 AOVs .....	213
1.11.2 BSDFs .....	216
1.11.3 Closures .....	222
1.11.3.1 Converting to Closures .....	226
1.11.4 Light Filters .....	228
1.11.5 Light Loops .....	232
1.11.6 Light Path Expression AOVs .....	236
1.11.7 OSL Shaders .....	242
1.11.8 Shader Data Type Conversions .....	249
1.11.9 Trace Sets .....	251
1.11.10 Vector Displacement .....	256
1.11.11 Volume Shaders .....	266
1.12 Textures .....	270
1.12.1 maketx .....	272
1.12.2 Filename Tokens .....	274
1.12.2.1 <attr:> .....	275
1.12.2.2 <tile> .....	276
1.12.2.3 <udim> .....	277
1.13 Third Party Shaders .....	278
1.14 Legal .....	279
1.14.1 End User License Agreement .....	280
1.15 System Requirements .....	282

1.16 Arnold Denoiser (Noise) .....	283
1.17 FAQs .....	286
1.17.1 Arnold 5.0 FAQ .....	287
1.17.2 General FAQ .....	289
2. API Release Notes .....	296
2.1 5.2.0.0 .....	298
2.2 5.1.1.2 .....	300
2.3 5.1.1.1 .....	303
2.4 5.1.1.0 .....	304
2.5 5.1.0.1 .....	306
2.6 5.1.0.0 .....	310
2.7 5.0.2.4 .....	314
2.8 5.0.2.3 .....	319
2.9 5.0.2.2 .....	324
2.10 5.0.2.1 .....	329
2.11 5.0.2.0 .....	333
2.12 5.0.1.5 .....	337
2.13 5.0.1.4 .....	338
2.14 5.0.1.3 .....	339
2.15 5.0.1.2 .....	340
2.16 5.0.1.1 .....	341
2.17 5.0.1.0 .....	342
2.18 5.0.0.3 .....	345
2.19 5.0.0.2 .....	346
2.20 5.0.0.1 .....	347
2.21 5.0.0.0 .....	348
2.22 4.2.16.0 .....	358
2.22.1 4.2.16.1 .....	360
2.22.2 4.2.16.2 .....	361
2.22.3 4.2.16.3 .....	362
2.22.4 4.2.16.4 .....	363
2.23 4.2.15.0 .....	364
2.23.1 4.2.15.1 .....	365
2.24 4.2.14.0 .....	366
2.24.1 4.2.14.1 .....	368
2.24.2 4.2.14.2 .....	369
2.24.3 4.2.14.3 .....	370
2.24.4 4.2.14.4 .....	371
2.25 4.2.13.0 .....	372
2.25.1 4.2.13.1 .....	374
2.25.2 4.2.13.2 .....	375
2.25.3 4.2.13.3 .....	376
2.25.4 4.2.13.4 .....	377
2.25.5 4.2.13.5 .....	378
2.25.6 4.2.13.6 .....	379
2.26 4.2.12.0 .....	380
2.26.1 4.2.12.1 .....	384
2.26.2 4.2.12.2 .....	385
2.26.3 4.2.12.3 .....	386
2.27 4.2.11.0 .....	387
2.27.1 4.2.11.1 .....	389
2.27.2 4.2.11.2 .....	390
2.27.3 4.2.11.3 .....	391
2.28 4.2.10.0 .....	392
2.28.1 4.2.10.1 .....	393
2.28.2 4.2.10.2 .....	394
2.29 4.2.9.0 .....	395
2.30 4.2.8.0 .....	397
2.31 4.2.7.0 .....	399
2.31.1 4.2.7.1 .....	401
2.31.2 4.2.7.2 .....	402
2.31.3 4.2.7.3 .....	403
2.31.4 4.2.7.4 .....	404
2.31.5 4.2.7.5 .....	405
2.32 4.2.6.0 .....	406
2.33 4.2.4.0 .....	408
2.34 4.2.3.0 .....	410
2.34.1 4.2.3.1 .....	412
2.35 4.2.2.0 .....	413
2.36 4.2.0.0 .....	416
2.37 4.2.1.2 .....	421
2.38 4.2.1.1 .....	422
2.39 4.2.1.0 .....	423
2.40 4.2.0.6 .....	425
2.41 4.1.3.5 .....	431
2.42 4.1.3.3 .....	433
2.43 4.1.3.2 .....	435

2.44 4.1.3.1	436
2.45 4.1.3.0	438
2.46 4.1.2.0	439
2.47 4.1.1.0	440
2.48 4.1.0.0	442
2.49 4.0.16.2	446
2.50 4.0.16.1	448
2.51 4.0.16.0	450
2.52 4.0.15.1	452
2.53 4.0.15.0	454
2.54 4.0.14.0	455
2.55 4.0.13.1	456
2.56 4.0.13.0	457
2.57 4.0.12.0	458
2.58 4.0.11.0	460
2.59 4.0.10.2	463
2.60 4.0.10.1	465
2.61 4.0.10.0	467
2.62 4.0.9.1	469
2.63 4.0.9.0	470
2.64 4.0.8.0	471
2.65 4.0.7.0	473
2.66 4.0.6.0	475
2.67 4.0.5.3	477
2.68 4.0.5.2	479
2.69 4.0.5.1	481
2.70 4.0.5.0	483
2.71 4.0.4.0	484
2.72 4.0.3.0	486
2.73 4.0.2.0	487
2.74 4.0.1.3	489
2.75 4.0.1.2	491
2.76 4.0.1.1	493
2.77 4.0.1.0	495
2.78 4.0.0.0	497
2.79 3.x	503
2.79.1 3.3.13.2	504
2.79.2 3.3.13.1	506
2.79.3 3.3.13.0	508
2.79.4 3.3.12.1	510
2.79.5 3.3.12.0	511
2.79.6 3.3.11.1	512
2.79.7 3.3.11.0	514
2.79.8 3.3.10.2	516
2.79.9 3.3.10.1	518
2.79.10 3.3.10.0	520
2.79.11 3.3.9.0	521
2.79.12 3.3.8.0	523
2.79.13 3.3.7.0	525
2.79.14 3.3.6.0	527
2.79.15 3.3.5.0	529
2.79.16 3.3.4.2	531
2.79.17 3.3.3.2	535
2.79.18 3.3.2.2	537
2.79.19 3.3.0.0	539
2.79.20 3.3.1.0	541
2.79.21 3.2.5.0	543
2.79.22 3.2.4.0	544
2.79.23 3.2.3.1	545
2.79.24 3.2.2.0	546
2.79.25 3.2.1.0	547
2.79.26 3.2.0.0	548
2.79.27 3.1.1.0	551
3. Archive / Updates	552

# Arnoldpedia

for Arnold 5

- [Introduction](#)
- [Technical Specifications](#)
- [Arnold 5 Porting Guide](#)
- [API Examples](#)
- [API FAQ](#)
- [Arnold Scene Source \(ass\)](#)
- [Command Line Rendering \(kick\)](#)
- [Displacement](#)
- [Plugins](#)
- [Rendering](#)
- [Shaders](#)
- [Textures](#)
- [Third Party Shaders](#)
- [Legal](#)
- [System Requirements](#)
- [Arnold Denoiser \(Noise\)](#)
- [FAQs](#)

# Introduction

Arnold is an advanced cross-platform rendering library, or API, developed by Solid Angle and used by a number of prominent organizations in film, television, and animation, including Sony Pictures Imageworks. It was developed as a photo-realistic, physically-based ray tracing alternative to traditional scanline based rendering software for CG animation.

Arnold uses cutting-edge algorithms that make the most effective use of your computer's hardware resources: memory, disk space, multiple processor cores, and SIMD/SSE units.

The Arnold architecture was designed to easily adapt to existing pipelines. It is built on top of a pluggable node system; users can extend and customize the system by writing new shaders, cameras, filters and output driver nodes, as well as procedural geometry, custom ray types and user-defined geometric data. The primary goal of the Arnold architecture is to provide a complete solution as a primary renderer for animation and visual effects. However, Arnold can also be used as:

- a ray server for traditional scanline renderers
- a tool for baking/procedural generation of lighting data (lightmaps for videogames)
- an interactive rendering and relighting tool

## Why is Arnold different?

Arnold is a highly optimized, unbiased, physically-based 'Monte Carlo' ray/path tracing engine. It doesn't use caching algorithms that introduce artifacts like photon mapping and final gather. It is designed to efficiently render the increasingly complex images demanded by animation and visual effects facilities while simplifying the pipeline, infrastructure requirements and user experience.

Arnold provides interactive feedback, often avoiding the need for many render passes and allowing you to match on-set lighting more efficiently. By removing many of the frustrating elements of other renderers, Arnold fits better with your work-flow, produces beautiful, predictable and bias-free results, and puts the fun back into rendering!

## What is wrong with algorithms like photon mapping or final gather?

Such algorithms attempt to cache data that can be re-sampled later, to speed up rendering. However, in doing so, they use up large amounts of memory, introduce intermediate steps that break interactivity, and introduce bias into the sampling that causes visual artifacts. They also require artists to understand the details of how these algorithms work to correctly choose various control settings to get any speed up at all without ruining the render. Worse than that, these settings are almost always affected by other things in the scene, so it's often possible to accidentally use settings for the cache creation/use that make things worse, not better, or that work fine in one situation but are terrible in another, seemingly similar, situation.

In short, they are not predictable, other than for very experienced users, and require artists to learn way too much about the algorithms to gain any benefit. At Solid Angle, we believe that your time is more valuable than your computer's time; why spend an extra 30 minutes working with photon mapping or final gather settings, even if it saves 30 minutes render time (and more often than not it doesn't). That's still 30 minutes not spent modeling, animating or lighting.

## What are the advantages of being physically-based? Does this have implications for modeling and lighting scenes?

The advantage of being physically-based is that artists can work in a physically accurate, high dynamic range work-flow, and by using plausible values (being consistent about how you set lighting intensities and modeling scale) the end results will be more predictable. The process becomes much more akin to how a real scene is lit and photographed. It also ensures that other aspects of rendering are not broken.

# Technical Specifications

## General

- Pure ray-tracing solution for film production rendering
- Unbiased, physically based uni-directional path tracer
- Cross-platform (Windows, Mac OS X, Linux)

## Pipeline

- Flexible and extensible node based architecture
- User data definition (primvars)
- C++ API (with Python bindings)
- Ray server capabilities

## Workflow

- Reduced number of parameters to tweak
- Progressive refinement
- Progressive modification of the scene elements
- Takes full advantage of multi-core CPUs and SSE units

## Primitives

- Support for extremely large polygon meshes with low memory footprint
- Analytically ray-traced curves in ribbon and thick (cylinder) modes. Optional screen-space width constraints at render time
- Points primitive for efficient rendering of hundreds of millions of particles
- Analytic primitives (box, cone, sphere, cylinder, disk, plane)
- Non-trimmed NURBS
- Implicit surfaces
- Volumes (including native support for OpenVDB)

## Subdivision and Displacement

- Catmull-Clark and linear subdivision
- Adaptive subdivision based on an arbitrary camera
- Vector Displacement
- Autobump

## Instances

- Optimized for large numbers of instances
- Transform and shading overrides

## Procedural Geometry

- Deferred creation of geometry nodes at render time
- Modular assembly of scenes (DSO, .ass, .ply, .obj)
- Recursive procedurals

## Cameras

- Perspective, fisheye, spherical, cylindrical, orthogonal, VR
- Bokeh effects
- Depth of Field
- API for writing custom cameras
- Filtermap for vignetting-like effects
- UV remapping for custom lens distortion

## Output Drivers

- Built-in drivers for common formats (JPEG, TIFF, PNG, EXR)
- Deep Image Data support (EXR)
- Arbitrary Output Variables (AOV) of any data type
- Per-AOV filtering

- API for custom drivers

## **Texture System**

- Based on a custom branch of OpenImageIO
- State-of-the-art anisotropic filtering
- On-demand loading of texture tiles
- In-memory cache for efficient rendering of huge texture data sets
- Low-latency texture lookups optimized for up to 256 threads
- Per-file texture checksums to avoid reloading files with identical pixel content

## **Motion Blur**

- Efficient Transformation and deformation motion blur
- All parameters can potentially support motion blur

## **Lights**

- High-quality area lights (quad, disk, cylinder, sphere, mesh, skydome, IES) with multiple importance sampling
- Image-based lighting in skydome and quad lights with importance sampling
- Light filters (attenuation, gobos, blockers...)
- Homogeneous and heterogeneous volumetric participating media

## **Shading**

- C++ API for programmable shaders
- OSL support for programmable shaders
- Physically-based BSDFs
- Procedural texturing primitives (Perlin noise, fractal noise, cellular)
- Message passing
- Pointcloud-less, ray-traced sub-surface scattering

## **Native scene description format (ASS)**

- Human readable, easy to tweak
- Standalone rendering via kick
- Pipeline friendly
- File concatenation, easy building of assemblies

## **Debugging**

- Debug modes as a native function of the renderer
- Complete rendering process log
- "Ignore" options and utility shaders

## **Licensing**

- Floating license scheme based on Reprise Software's RLM

# Arnold 5 Porting Guide

The Arnold 5 porting guide is available to download below.

File	Modified
Arnold5PortingGuide.pdf	May 31, 2017 by Lee Griggs
Drag and drop to upload or <a href="#">browse for files</a>	

## API Examples

- Converting to Closures
- Creating and Rendering a Scene
- Custom Gaussian Filter
- Iterate an Arnold Array With the Python Bindings
- Updating and Re-Rendering With Python
- Using the PointCloud API
- Writing a Camera Node
- Writing a Driver
- Writing a Light Filter
- Writing a Raw Driver
- Procedurals
- Shaders (API)

## Creating and Rendering a Scene

This example C++ code uses the Arnold API to create a basic scene containing a sphere, a textured quad mesh, a point light and a camera. It will then render the scene to a user-specified JPEG file on disk.

### Source Code

#### example.cpp

```
#include <ai.h>

int main()
{
    // start an Arnold session, log to both a file and the console
    AiBegin();
    AiMsgSetLogFileName( "scenel.log" );
    AiMsgSetConsoleFlags(AI_LOG_ALL);

    // create a sphere geometric primitive
    AtNode *sph = AiNode("sphere");
    AiNodeSetStr(sph, "name", "mysphere");
    AiNodeSetVec(sph, "center", 0.0f, 4.0f, 0.0f);
    AiNodeSetFlt(sph, "radius", 4.0f);

    // create a polymesh, with UV coordinates
    AtNode *mesh = AiNode("polymesh");
    AiNodeSetStr(mesh, "name", "mymesh");
    AtArray* nsides_array = AiArray(1, 1, AI_TYPE_UINT, 4);
    AiNodeSetArray(mesh, "nsides", nsides_array);
    AtArray* vlist_array = AiArray(12, 1, AI_TYPE_FLOAT, -10.f, 0.f, 10.f, 10.f,
0.f, 10.f, -10.f, 0.f, -10.f, 10.f, 0.f, -10.f);
    AiNodeSetArray(mesh, "vlist", vlist_array);
    AtArray* vidxs_array = AiArray(4, 1, AI_TYPE_UINT, 0, 1, 3, 2);
    AiNodeSetArray(mesh, "vidxs", vidxs_array);
    AtArray* uvlist_array = AiArray(8, 1, AI_TYPE_FLOAT, 0.f, 0.f, 1.f, 0.f, 1.f,
1.f, 0.f, 1.f);
    AiNodeSetArray(mesh, "uvlist", uvlist_array);
    AtArray* uvidxs_array = AiArray(4, 1, AI_TYPE_UINT, 0, 1, 2, 3);
    AiNodeSetArray(mesh, "uvidxs", uvidxs_array);

    // create a red standard surface shader
    AtNode *shader1 = AiNode("standard_surface");
    AiNodeSetStr(shader1, "name", "myshader1");
    AiNodeSetRGB(shader1, "base_color", 1.0f, 0.02f, 0.02f);
    AiNodeSetFlt(shader1, "specular", 0.05f);

    // create a textured standard surface shader
    AtNode *shader2 = AiNode("standard_surface");
    AiNodeSetStr(shader2, "name", "myshader2");
    AiNodeSetRGB(shader2, "base_color", 1.0f, 0.0f, 0.0f);

    // create an image shader for texture mapping
    AtNode *image = AiNode("image");
    AiNodeSetStr(image, "name", "myimage");
    AiNodeSetStr(image, "filename", "solidangle_icon.png");
```

```

AiNodeSetFlt(image, "sscale", 4.f);
AiNodeSetFlt(image, "tscale", 4.f);
// link the output of the image shader to the color input of the surface shader
AiNodeLink(image, "base_color", shader2);

// assign the shaders to the geometric objects
AiNodeSetPtr(sph, "shader", shader1);
AiNodeSetPtr(mesh, "shader", shader2);

// create a perspective camera
AtNode *camera = AiNode("persp_camera");
AiNodeSetStr(camera, "name", "mycamera");
// position the camera (alternatively you can set 'matrix')
AiNodeSetVec(camera, "position", 0.f, 10.f, 35.f);
AiNodeSetVec(camera, "look_at", 0.f, 3.f, 0.f);
AiNodeSetFlt(camera, "fov", 45.f);

// create a point light source
AtNode *light = AiNode("point_light");
AiNodeSetStr(light, "name", "mylight");
// position the light (alternatively use 'matrix')
AiNodeSetVec(light, "position", 15.f, 30.f, 15.f);
AiNodeSetFlt(light, "intensity", 4500.f); // alternatively, use 'exposure'
AiNodeSetFlt(light, "radius", 4.f); // for soft shadows

// get the global options node and set some options
AtNode *options = AiUniverseGetOptions();
AiNodeSetInt(options, "AA_samples", 8);
AiNodeSetInt(options, "xres", 480);
AiNodeSetInt(options, "yres", 360);
AiNodeSetInt(options, "GI_diffuse_depth", 4);
// set the active camera (optional, since there is only one camera)
AiNodeSetPtr(options, "camera", camera);

// create an output driver node
AtNode *driver = AiNode("driver_jpeg");
AiNodeSetStr(driver, "name", "mydriver");
AiNodeSetStr(driver, "filename", "scenel.jpg");

// create a gaussian filter node
AtNode *filter = AiNode("gaussian_filter");
AiNodeSetStr(filter, "name", "myfilter");

// assign the driver and filter to the main (beauty) AOV,
// which is called "RGBA" and is of type RGBA
AtArray *outputs_array = AiArrayAllocate(1, 1, AI_TYPE_STRING);
AiArraySetStr(outputs_array, 0, "RGBA RGBA myfilter mydriver");
AiNodeSetArray(options, "outputs", outputs_array);

// finally, render the image!
AiRender(AI_RENDER_MODE_CAMERA);

// ... or you can write out an .ass file instead
//AiASSWrite("scenel.ass", AI_NODE_ALL, FALSE);

// Arnold session shutdown
AiEnd();

```

```
    return 0;  
}
```

## Compiling

The following commands compile and run the example on various platforms:

### Linux

```
export ARNOLD_PATH=/path/to/arnold  
c++ example.cpp -o example -Wall -O2 -I$ARNOLD_PATH/include -L$ARNOLD_PATH/bin -lai  
.example
```

### macOS

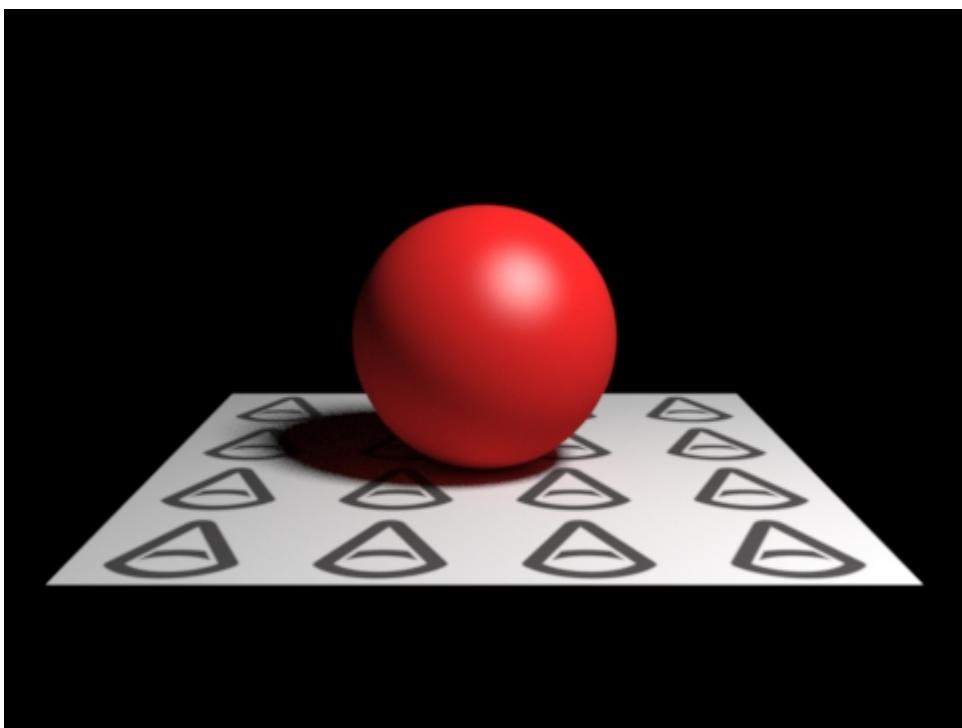
```
export ARNOLD_PATH=/path/to/arnold  
c++ example.cpp -o example -Wall -O2 -I$ARNOLD_PATH/include -L$ARNOLD_PATH/bin -lai  
.example
```

### Windows Visual Studio command prompt

```
set ARNOLD_PATH=c:/path/to/arnold  
cl example.cpp /I %ARNOLD_PATH%/include %ARNOLD_PATH%/lib/ai.lib /link  
/out:example.exe  
example
```

## Output

Once rendered, the output should look like this:



## Custom Gaussian Filter

This is example code of a custom gaussian filter.

```
#include <ai.h>
#include <string.h>

AI_FILTER_NODE_EXPORT_METHODS(CustomGaussianFilterMtd);

static AtString WIDTH( "width" );

node_parameters
{
    AiParameterFlt("width", 2.0f);
}

node_initialize
{
    AiFilterInitialize(node, false, NULL, NULL);
}

node_update
{
    AiFilterUpdate(node, AiNodeGetFlt(node, WIDTH));
}

node_finish
{
    AiFilterDestroy(node);
}

filter_output_type
{
    switch (input_type)
    {
        case AI_TYPE_RGBA:
            return AI_TYPE_RGBA;
        default:
            return AI_TYPE_NONE;
    }
}

filter_pixel
{
    const float width = AiNodeGetFlt(node, WIDTH);

    float aweight = 0.0f;
    AtRGBA avalue = AI_RGBA_ZERO;

    while (AiAOVSampleIteratorGetNext(iterator))
    {
        // take into account adaptive sampling
        float inv_density = AiAOVSampleIteratorGetInvDensity(iterator);
        if (inv_density <= 0.f)
            continue;
        else
            avalue += AiAOVSampleIteratorGetColor(iterator) * inv_density;
    }

    avalue /= width;
    AiNodeSetFlt(node, WIDTH, avalue);
}
```

```

// determine distance to filter center
const AtPoint2& offset = AiAOVSampleIteratorGetOffset(iterator);
const float r = SQR(2 / width) * (SQR(offset.x) + SQR(offset.y));
if (r > 1.0f)
    continue;
// gaussian filter weight
const float weight = fast_exp(2 * -r) * inv_density;

// accumulate weights and colors
avalue += weight * AiAOVSampleIteratorGetRGBA(iterator);
aweight += weight;
}

// compute final filtered color
if (aweight != 0.0f)
    avalue /= aweight;

*((AtRGBA*)data_out) = avalue;
}

node_loader
{
    if (i>0)
        return false;

    node->methods      = CustomGaussianFilterMtd;
    node->output_type   = AI_TYPE_NONE;
    node->name         = "custom_gaussian_filter";
    node->node_type     = AI_NODE_FILTER;
    strcpy(node->version, AI_VERSION);
}

```

```
    return true;  
}
```

## Iterate an Arnold Array With the Python Bindings

The following snippet code opens the "test.ass" file attached in this wiki, find the object named "mesh", get the shader array, and print a list of its shaders.

```
#!/usr/bin/env python
from arnold import *

AiBegin()
AiASSLoad("test.ass")

mesh = AiNodeLookUpByName('mesh')
shaders = AiNodeGetArray(mesh, "shader");
for i in range(AiArrayGetNumElements(shaders)):
    shader = cast(AiArrayGetPtr(shaders, i), POINTER(AtNode))
    print AiNodeGetName(shader.contents)

AiEnd()
```

## Updating and Re-Rendering With Python

### ***Re-rendering with updated cameras in a sequence with Python***

This tutorial shows how to use Python to update only the camera in a static scene and re-render without re-cooking geometry.

In some scenes, there are large geometry datasets or procedurals that take significant time to load but that do not change from frame to frame. In these cases, it is nice to take advantage of this and not be forced to reload or rebuild the geometry. Arnold allows us to update or add nodes, and the raytracing acceleration data structures are not view-dependent, which is why IPR re-renders are so fast. We can take advantage of this.

If you have a static scene with an animating camera, you will need to update two nodes every frame: the camera and the output driver. Obviously, if you want to render from a new location, you need to have the new camera. The output driver's filename needs to be updated because, if you used the same exact driver, you would overwrite the frame you had previously rendered.

In the script below, the Python function **AiASSLoad(assfile, AI\_NODE\_ALL)** reads in everything from an ass file. We call this for the first frame we render, but we don't want subsequent frames to re-read in our static geometry. We do this by reading in only the camera, driver, filter, and option nodes using **AiASSLoad(assdir+ass, AI\_NODE\_CAMERA | AI\_NODE\_DRIVER | AI\_NODE\_FILTER | AI\_NODE\_OPTIONS)**.

Arnold is happy to have as many cameras drivers and filters as a user chooses to put into a scene, but we want to replace these nodes with the new ones. To do this, we just loop through and delete all cameras, drivers, and filters. That leaves our scene ready for the new ones we will read into the scene. Then, after we read them in, we need to make a list of them, so we can easily delete them later for the next frame.

Finally, we simply render the frame and loop back. This method was used to render the Mandelbulb sequence for the [Managing RAM and threading in procedurals](#) tutorial. A high-resolution Mandelbulb takes about 45 minutes to compute, but only 4 minutes to render. This was a perfect test for this Python script, and there is a ten frame sequence of .ass files attached for you to test this script with.

camera\_only.py

```
#!/usr/bin/env python

from arnold import *
import os
import sys

if len(sys.argv)!=2:
    print "\nusage: camera_only.py path/to/ass/files/directory/"
    print "the directory should contain only the sequence of assfiles you want
rendered....\n"
    sys.exit()
# this should be a directory full of .ass files
assdir = sys.argv[1]
# begin, load first frame, and render it.
AiBegin()
# use sorted() to get them in sequence
assfiles = sorted(os.listdir(assdir))
# first time, load ALL nodes
AiASSLoad (os.path.join(assdir, assfiles[0]), AI_NODE_ALL)
# and render your first frame.
AiRender()

# list to hold cameras, filters, and drivers.
deletion_list = []

# this will let you iterate over all camera, driver, and filter nodes in the Arnold
universe
nodeIter = AiUniverseGetNodeIterator(AI_NODE_CAMERA | AI_NODE_DRIVER |
AI_NODE_FILTER)
while not AiNodeIteratorFinished(nodeIter):
    node = AiNodeIteratorGetNext(nodeIter)
    deletion_list.append(node)
```

```
AiNodeIteratorDestroy(nodeIter)
# you have already rendered the first frame
del assfiles[0]
# now cycle through all frames in the given directory
for ass in assfiles:
    # echo out useful information
    print "loading "+ass
    for node in deletion_list:
        AiNodeDestroy(node)
    # this loads only cameras, drivers, filters, and an option node.
    AiASSLoad (os.path.join(assdir, ass), AI_NODE_CAMERA | AI_NODE_DRIVER |
AI_NODE_FILTER | AI_NODE_OPTIONS)
    # clear our old list
    deletion_list = []
    #loop through nodes making a fresh list of cameras, filters, and drivers
    nodeIter = AiUniverseGetNodeIterator(AI_NODE_CAMERA | AI_NODE_DRIVER |
AI_NODE_FILTER)
    while not AiNodeIteratorFinished(nodeIter):
        node = AiNodeIteratorGetNext(nodeIter)
        deletion_list.append(node)
    AiNodeIteratorDestroy(nodeIter)
    # finally, render this frame before looping again
```

```

    AiRender()
# we are done, this is our final call
AiEnd()

```

This type of script was used to render this sequence: [Mandelbulb](#). The quarter billion spheres took significant time to generate, but the re-renders were quite fast.

## Re-rendering after updating the vertices of a quad area light

This second example needs the multilight\_area.ass file attached. It will first render the scene, then update the vertices of the area light called "light\_0003", and then re-render.

```

from arnold import *

AiBegin()
AiMsgSetConsoleFlags(AI_LOG_ALL);
AiASSLoad("multilight_area.ass")

print("***** 1st pass, render.tif *****")

AiRender()

print("***** 2nd pass, render2.tif *****")

# Change output filename
driver = AiNodeLookUpByName('testrender')
if driver is not None and AiNodeIs(driver, 'driver_tiff') == True:
    AiNodeSetStr(driver, 'filename', 'render2.tif')

# Muck with the blue quad light
quadlight = AiNodeLookUpByName('light_0003')
if quadlight is not None and AiNodeIs(quadlight, 'quad_light') == True:
    verts = AiNodeGetArray(quadlight, 'vertices')
    AiArraySetPnt(verts, 0, AiArrayGetPnt(verts, 0) + AtPoint(0, 0, -2))
    AiArraySetPnt(verts, 1, AiArrayGetPnt(verts, 1) + AtPoint(0, 0, -2))
    AiArraySetPnt(verts, 2, AiArrayGetPnt(verts, 2) + AtPoint(0, 0, -2))
    AiArraySetPnt(verts, 3, AiArrayGetPnt(verts, 3) + AtPoint(0, 0, -2))
    #v0 = AiArrayGetPnt(verts, 0)
    #print("0: %f %f %f" % (v0.x, v0.y, v0.z))
    #v1 = AiArrayGetPnt(verts, 1)
    #print("1: %f %f %f" % (v1.x, v1.y, v1.z))
    #v2 = AiArrayGetPnt(verts, 2)
    #print("2: %f %f %f" % (v2.x, v2.y, v2.z))
    #v3 = AiArrayGetPnt(verts, 3)
    #print("3: %f %f %f" % (v3.x, v3.y, v3.z))

AiRender()

AiEnd()

```

## Using the PointCloud API

### **SSS point cloud iterator**

This test computes irradiance at the points provided through the iterative point cloud API.

```
#include "ai.h"

AI_SHADER_NODE_EXPORT_METHODS(PointCloudShaderMethods);

enum PointCloudShaderParam
{
    p_searchRadius
};

node_parameters
{
    AiParameterFLT("search_radius", 0.02f);
}

struct PointCloudSample
{
    AtPoint point;
    AtRGB irradiance;
};

struct PointCloudShaderData
{
    AtCritSec cs;
    unsigned int sampleCount;
    PointCloudSample* samples;
};

void ComputeIrradiance(AtShaderGlobals* sg, PointCloudShaderData* data)
{
    // Allocate sample storage
    const unsigned int sampleCount = AiPointCloudGetSampleCount(sg);
    PointCloudSample* samples = (PointCloudSample*) AiMalloc(sampleCount *
sizeof(PointCloudSample));

    // Initialize loop variables
    AtPointCloudIterator* iter = AiPointCloudIteratorCreate(sg);
    unsigned int i = 0;

    // Loop through samples
    while (!AiPointCloudIteratorFinished(iter))
    {
        AtPointCloudSample sample = AiPointCloudIteratorGetNext(iter);
        samples[i].point = sample.world_position;
        samples[i].irradiance = AiSSSEvaluateIrradiance(sg, sample.uv.x, sample.uv.y,
sample.face, &sample.world_position, i);
        i++;
    }

    // Clean up and store results
    // Note that data->samples should be assigned only after samples have been fully
```

```

computed
{
    AiPointCloudIteratorDestroy(iter);
    data->samples = samples;
    data->sampleCount = sampleCount;
}

node_initialize
{
    PointCloudShaderData* data = (PointCloudShaderData*)
AiMalloc(sizeof(PointCloudShaderData));
    AiNodeSetLocalData(node, data);
    AiCritSecInitRecursive(&(data->cs));
    data->sampleCount = 0;
    data->samples = NULL;
}

node_update
{
}

node_finish
{
    if (AiNodeGetLocalData(node))
    {
        PointCloudShaderData* data = (PointCloudShaderData*)
AiNodeGetLocalData(node);
        AiCritSecClose(&(data->cs));
        AiFree(data->samples);
        AiFree((void*) data);
        AiNodeSetLocalData(node, NULL);
    }
}

shader_evaluate
{
    sg->out.RGB = AI_RGB_BLACK;

    PointCloudShaderData* data = (PointCloudShaderData*) AiNodeGetLocalData(node);

    if (!data->samples)
    {
        AiCritSecEnter(&(data->cs));
        if (!data->samples)
            ComputeIrradiance(sg, data);
        AiCritSecLeave(&(data->cs));
    }

    // simple brute force sample lookup
    if (static_cast<volatile PointCloudSample*>(data->samples))
    {
        for (unsigned int i = 0; i < data->sampleCount; ++i)
        {
            const float searchRadius = AiShaderEvalParamFlt(p_searchRadius);
            if (AiV3Length(data->samples[i].point - sg->P) < searchRadius)
            {
                sg->out.RGB = data->samples[i].irradiance;
                break;
            }
        }
    }
}

```

```
        }
    }
}

node_loader
{
    if (i > 0)
        return FALSE;

    node->methods = PointCloudShaderMethods;
    node->output_type = AI_TYPE_RGB;
    node->name = "point_cloud_shader";
    node->node_type = AI_NODE_SHADER;
    strcpy(node->version, AI_VERSION);
```

```
    return TRUE;  
}
```



## Writing a Camera Node

Here is the source code for a simple perspective camera node with vignetting and lens distortion:

```
#include <ai.h>
#include <string.h>

AI_CAMERA_NODE_EXPORT_METHODS(MyCameraMethods)

#define _fov (params[0].FLT)

node_parameters
{
    AiParameterFlt("fov", 60.f);
}

node_initialize
{
    AiCameraInitialize(node, NULL);
}

node_update
{
    AiCameraUpdate(node, false);
}

node_finish
{
    AiCameraDestroy(node);
}

camera_create_ray
{
    const AtParamValue* params = AiNodeGetParams(node);
    float tan_fov = tanf(_fov * AI_DTOR / 2);

    AtPoint p = { input->sx * tan_fov, input->sy * tan_fov, 1 };

    // warp ray origin with a noise vector
    AtVector noise_point = { input->sx, input->sy, 0.5f };
    noise_point *= 5;
    AtVector noise_vector = AivNoise3(noise_point, 1, 0.f, 1.92f);
    output->origin = noise_vector * 0.04f;
    output->dir = Aiv3Normalize(p - output->origin);

    // vignetting
    float dist2 = input->sx * input->sx + input->sy * input->sy;
    output->weight = 1 - dist2;

    // now looking down -Z
    output->dir.z *= -1;
}

node_loader
{
    if (i != 0) return false;
```

```
node->methods      = MyCameraMethods;
node->output_type  = AI_TYPE_UNDEFINED;
node->name         = "mycamera";
node->node_type    = AI_NODE_CAMERA;
```

```

    strcpy(node->version, AI_VERSION);
    return true;
}

```

In camera\_create\_ray you also need to compute the direction and position derivatives. This is important for correct filtering.

For a perspective camera the position is always the same, but the direction changes from pixel to pixel. Follows some sample code to compute the derivatives for a perspective camera (we do not take into account uv noise):

```

float fov = AiArrayInterpolateFlt(_fov, input->relative_time, 0);
fov *= (float) (AI_DTOR * 0.5);
float tan_fov = tanf(fov);

...
// scale derivatives
float dsx = input->dsx * tan_fov;
float dsy = input->dsy * tan_fov;
...

AtVector d = p; // direction vector == point on the image plane
double d_dot_d = AiV3Dot(d, d);
double temp = 1.0 / sqrt(d_dot_d * d_dot_d * d_dot_d);

// already initialized to 0's, only compute the non zero coordinates
output->dDdx.x = (d_dot_d * dsx - (d.x * dsx) * d.x) * temp;
output->dDdx.y = (- (d.x * dsx) * d.y) * temp;
output->dDdx.z = (- (d.x * dsx) * d.z) * temp;
output->dDdy.x = (- (d.y * dsy) * d.x) * temp;
output->dDdy.y = (d_dot_d * dsy - (d.y * dsy) * d.y) * temp;
output->dDdy.z = (- (d.y * dsy) * d.z) * temp;

// output->dOd* is also initialized to 0s, the correct value

```

Last modified 22 months ago

## Writing a Driver

### *Writing an output driver*

An output driver gives you access to the one filtered value per pixel, after a bucket has been completely rendered. You can use drivers for example to write to new image file formats, or to send pixels to a display device, to an application's window, etc.

You can specify a driver in an .ass file like this:

```
outputs 1 1 STRING
"RGBA RGBA my_main_filter my_main_driver"
```

You can also specify multiple drivers for different AOVs, potentially using different filters and data types:

```
outputs 3 1 STRING
"RGBA RGBA my_filter my_main_driver"
"direct_diffuse RGBA my_filter my_direct_diffuse_driver"
"indirect_diffuse RGBA my_filter my_indirect_diffuse_driver"
```

The implementation of a sample driver that writes out to a file a list of all the objects in a pointer AOV would look like:

```
#include <ai.h>
#include <strings.h>
#include <fstream>
#include <unordered_map>

// This driver will write to a file a list of all the objects in a pointer AOV

AI_DRIVER_NODE_EXPORT_METHODS(DriverPtrMtd);
namespace ASTR {
    const AtString name("name");
    const AtString filename("filename");
};

typedef struct {
    std::unordered_map<AtString, AtNode*, AtStringHash> names;
} DriverPtrStruct;

node_parameters
{
    AiParameterSTR(ASTR::filename, "objects.txt");
}

node_initialize
{
    DriverPtrStruct *driver = new DriverPtrStruct();
    // initialize the driver
    AiDriverInitialize(node, false, driver);
}
```

```

driver_needs_bucket
{
    return true;
}

driver_process_bucket
{ }

node_update
{ }

driver_supports_pixel_type
{
    // this driver will only support pointer formats
    return pixel_type == AI_TYPE_POINTER || pixel_type == AI_TYPE_NODE;
}

driver_open
{ // this driver is unusual and happens to do all the writing at the end, so this
function is
    // empty.
}

driver_extension
{
    static const char *extensions[] = { "txt", NULL };
    return extensions;
}

driver_prepare_bucket
{ }

driver_write_bucket
{
    DriverPtrStruct *driver = (DriverPtrStruct *)AiDriverGetLocalData(node);
    const void *bucket_data;
    // Iterate over all the AOVs hooked up to this driver
    while (AiOutputIteratorGetNext(iterator, NULL, NULL, &bucket_data))
    {
        for (int y = 0; y < bucket_size_y; y++)
        {
            for (int x = 0; x < bucket_size_x; x++)
            {
                // Get source bucket coordinates for pixel
                int sidx = y * bucket_size_x + x;
                // Because of driver_supports_pixel_type, we know pixel is a
                // pointer to an AtNode.
                AtNode* pixel_node = ((AtNode **)bucket_data)[sidx];
                const AtString name = AiNodeGetStr(pixel_node, ASTR::name);
                driver->names.emplace(name, pixel_node);
            }
        }
    }
}

driver_close
{
}

```

```
DriverPtrStruct *driver = (DriverPtrStruct *)AiDriverGetLocalData(node);
std::ofstream myfile(AiNodeGetStr(node, ASTR::filename));
for (auto &i : driver->names)
    myfile << i.first << ":\t " <<i.second << std::endl;
myfile.close();
}

node_finish
{
    // Free local data
    DriverPtrStruct *driver = (DriverPtrStruct *)AiDriverGetLocalData(node);
    delete driver;
    AiDriverDestroy(node);
}

node_loader
{
    if (i>0)
        return false;
    node->methods = (AtNodeMethods*) DriverPtrMtd;
    node->output_type = AI_TYPE_NONE;
    node->name = "driver_ptr";
    node->node_type = AI_NODE_DRIVER;
```

```
    strcpy(node->version, AI_VERSION);
    return true;
}
```

## Writing a Light Filter

Light filters are arbitrary shaders that can modify, or filter, the incoming illumination from a given light. Light filters are just regular shaders attached to a light's "filters" parameter. Compared to an old-style RenderMan light shader, the only thing that an Arnold light filter cannot do is modify the light emitting geometry and its associated sampling.

Arnold currently ships with four built-in light filter shaders, which are explained in detail in the [MtoA docs](#). Here is an example using the built-in gobo shader:

```
image
{
    name myimage
    filename polkadots.jpg
}

gobo
{
    name mygobo
    slidemap mytexture
    rotate 30
    scale_s 2
    scale_t 2
}

spot_light
{
    name myspot
    ...
    filters mygobo
    # you can also attach a stack of filters:
    # filters 3 1 NODE gobol gobo2 gobo3
}
```

But you can also write your own light filter. Basically you can use some `shaderglobals` inputs (such as the light direction `sg->Ld`, the distance `sg->Ldist`, or, for a spot light, `sg->u/v`) and overwrite the unoccluded light intensity `sg->Liu` in-place. Below is a simple example that modulates the light intensity with a Perlin noise procedural texture:

```

#include <ai.h>
#include <strings.h>
AI_SHADER_NODE_EXPORT_METHODS(SimpleLightFilterMethods);
node_parameters { }
node_initialize { }
node_update { }
node_finish { }

shader_evaluate
{
    AtPoint2 p = {sg->u, sg->v};
    sg->Liu *= (AiPerlin2(p * 50) + 0.5f);
}

node_loader
{
    if (i > 0)
        return false;
    node->methods      = SimpleLightFilterMethods;
    node->output_type  = AI_TYPE_RGB;
    node->name          = "simple_light_filter";
    node->node_type     = AI_NODE_SHADER;
    strcpy(node->version, AI_VERSION);
    return true;
}

```

### simple light filter.ass

[Expand source](#)

```

options
{
    AA_samples 5
    GI_diffuse_depth 2
    GI_diffuse_samples 3
    background mysky
}
persp_camera
{
    name mycamera
    position 5 3.0 5
    look_at 0 0.8 1
    up 0 1 0
    fov 30
}
sky
{
    name mysky
    color 0.8 0.9 1.0
    intensity 0.5
}
lambert
{
    name mylambert
    Kd 0.5
}

```

```
}
```

```
plane
```

```
{
```

```
    name myplane
```

```
    normal 0 1 0
```

```
    shader mylambert
```

```
}
```

```
sphere
```

```
{
```

```
    name mysphere1
```

```
    center 0 1 0
```

```
    radius 1
```

```
    shader mylambert
```

```
}
```

```
sphere
```

```
{
```

```
    name mysphere2
```

```
    center 0 1 2
```

```
    radius 1
```

```
    shader mylambert
```

```
}
```

```
spot_light
```

```
{
```

```
    name mylight
```

```
    look_at 0 1 0
```

```
    position 4 5 1
```

```
    intensity 3.141
```

```
    color 1.0 0.7 0.1
```

```
    cone_angle 65
```

```
    penumbra_angle 2
```

```
    exposure 6
```

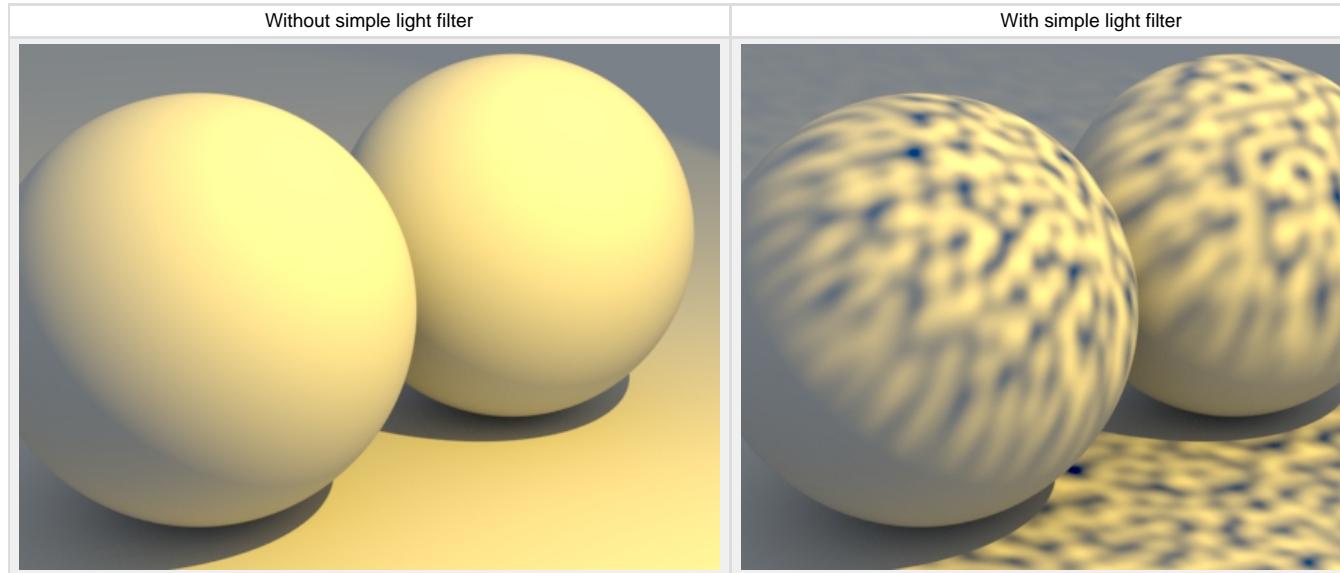
```
    filters my_light_filter
```

```
}
```

```
simple_light_filter
```

```
{  
    name my_light_filter  
}
```

The above example produces the following image:



## Writing a Raw Driver

In contrast to a regular output driver, which processes already filtered pixels, a "raw" driver gives you access to all the individual, unfiltered shading points along a ray, after a bucket has been completely rendered.

You can specify a raw driver in an .ass file like this:

```
outputs 2 1 STRING                                // this example has two drivers:  
    "RGBA RGBA my_main_filter my_main_driver"    // 1) regular driver for main RGBA  
    output and its filter  
    "raw_driver_name"                            // 2) a raw driver
```

Note that you use a raw driver just by specifying its name; there is no filtering.

The implementation of a sample raw driver would look like this:

```
#include <ai.h>  
  
AI_DRIVER_NODE_EXPORT_METHODS(DriverRAWMtd);  
  
typedef struct {  
    ...  
} DriverRAWStruct;  
  
node_loader  
{  
    if (i>0)  
        return FALSE;  
    node->methods = (AtNodeMethods*) DriverRAWMtd;  
    node->output_type = AI_TYPE_NONE;  
    node->name = "driver_raw";  
    node->node_type = AI_NODE_DRIVER;  
    strcpy(node->version, AI_VERSION);  
    return TRUE;  
}  
  
node_parameters  
{  
    AiParameterSTR( "filename", "deep.raw" );  
}  
  
node_initialize  
{  
    DriverRAWStruct *raw = new DriverRAWStruct();  
  
    static const char *required_aovs[] = { "FLOAT Z", "FLOAT A", NULL };  
  
    // Initialize the driver (set the required AOVs and indicate that  
    // we want values at all the depths)  
    AiRawDriverInitialize(node, required_aovs, true /* deep driver */, raw);  
}  
  
node_update
```

```

{
    ...
}

driver_supports_pixel_type
{
    // This function is not needed for raw drivers
    return TRUE;
}

driver_open
{
    ...
}

driver_extension
{
    static const char *extensions[] = { "raw", NULL };
    return extensions;
}

driver_needs_bucket
{
    return true;
}

driver_prepare_bucket
{
    ...
}

driver_process_bucket
{
    ...
}

driver_write_bucket
{
    DriverRAWStruct *raw = (DriverRAWStruct *)AiDriverGetLocalData(node);
    ...

    for (y = bucket_yo; y < bucket_yo + bucket_size_y; y++)
    {
        for (x = bucket_xo; x < bucket_xo + bucket_size_x; x++)
        {
            // Iterator for samples in this pixel
            AiAOVSampleIteratorInitPixel(sample_iterator, x, y);
            while (AiAOVSampleIteratorGetNext(sample_iterator))
            {
                AtPoint2 position = AiAOVSampleIteratorGetOffset(sample_iterator);
                while (AiAOVSampleIteratorGetNextDepth(sample_iterator))
                {
                    float a = AiAOVSampleIteratorGetAOVFlt(sample_iterator, "A");
                    float z = AiAOVSampleIteratorGetAOVFlt(sample_iterator, "Z");
                    ...
                }
            }
        }
    }
}

```

```
        }
    }
    ...
}

driver_close
{
    ...
}

node_finish
{
```

```
    ...
AiDriverDestroy(node);
}
```

## Procedurals

- Implementing a Random Pointcloud Procedural

## Implementing a Random Pointcloud Procedural

### ***Random Flake Procedural***

This is an example of making a geometry-generating procedural for Arnold. There are 2 files in this example; the code for the procedural, and an .ass scene file.

This procedural makes a random pointcloud with fractal distribution in space. There are 5 controls for the generation of the cloud:

- count: the number of points per recursion to generate
- recursions: the number of recursive steps to descend
- flake\_radius: the radius of the little spheres that make up the cloud
- cloud\_radius: the overall radius of the pointcloud
- seed: a random seed to generate a unique random cloud

The algorithm works like this: given a spherical region of radius R, scatter N points into it. For each point, scatter N random points within a radius of 1/2 the previous R, and so on, for X number of recursions, then hang a little sphere of radius S on the resulting points.

Warning: this algorithm creates poorly bound geometry, and is likely to build a few points outside the bounding box.

This is a sample render of a pointcloud with 2,015,539 points:



Here's the procedural code:

```
#include "ai.h"
#include <sstream>
#include <iostream>
#include <cstring>
#include <cstdlib>
```

```

using namespace std;

// global variables for procedural
int G_npoints;
float G_cloudrad;
float G_sphererad;
int G_recursions;
int G_counter;
int G_computedtotal;

// returns a random float between 0 and 1
static float myFrand()
{
    return (float)rand() / (float)RAND_MAX;
}

// returns a random vector in a unit sphere with a
// power function to bias it towards the center
static AtVector randvec(float power)
{
    AtVector out;
    out.x = myFrand() - 0.5;
    out.y = myFrand() - 0.5;
    out.z = myFrand() - 0.5;
    out = AiV3Normalize(out) * pow(myFrand(), power);
    return out;
}

// recursive function that creates random cloud with fractal clumping
static void makecloud(AtArray *pointarr, AtArray *radarr, AtVector cenIn, float
rad, int recursions)
{
    for (int i = 0; i < G_npoints; i++)
    {
        AtVector cen = randvec(0.5) * rad;
        AiArraySetPnt(pointarr, G_counter, cen+cenIn);
        AiArraySetFlt(radarr, G_counter, G_sphererad);
        G_counter++;
        if (recursions > 1)
            makecloud(pointarr, radarr, cen + cenIn, rad * 0.5, recursions - 1);
    }
}

static int MyInit(AtNode *mynode, void **user_ptr)
{
    *user_ptr = mynode; // make a copy of the parent procedural
    srand(AiNodeGetInt(mynode, "G_seed"));
    G_npoints = AiNodeGetInt(mynode, "G_npoints");
    G_sphererad = AiNodeGetFlt(mynode, "G_sphererad");
    G_cloudrad = AiNodeGetFlt(mynode, "G_cloudrad") - G_sphererad;
    G_recursions = AiNodeGetInt(mynode, "G_recursions");
    int npoints = 0;
    for (int i = 0; i < G_recursions; i++)
        npoints += pow(G_npoints, i);
    AiMsgInfo("[randflake] NPOINTS: %d", npoints);
    G_computedtotal = npoints;
}

```

```

    G_counter = 0;
    return TRUE;
}

static int MyCleanup(void *user_ptr)
{
    return TRUE;
}

static int MyNumNodes(void *user_ptr)
{
    return 1;
}

static AtNode *MyGetNode(void *user_ptr, int i)
{
    AtArray *pointarray = AiArrayAllocate(G_computedtotal, 1, AI_TYPE_POINT);
    AtArray *radiusarray = AiArrayAllocate(G_computedtotal, 1, AI_TYPE_FLOAT);
    AtVector center;
    center.x = center.y = center.z = 0;
    makecloud(pointarray, radiusarray, center, G_cloudrad, G_recursions - 1);

    AtNode *node = AiNode("points");
    AiNodeSetArray(node, "points", pointarray);
    AiNodeSetArray(node, "radius", radiusarray);
    AiNodeSetStr(node, "mode", "sphere");

    return node;
}

// vtable passed in by proc_loader macro define
proc_loader
{
    vtable->Init      = MyInit;
    vtable->Cleanup   = MyCleanup;
    vtable->NumNodes  = MyNumNodes;
    vtable->GetNode   = MyGetNode;
    strcpy(vtable->version, AI_VERSION);
}

```

```
    return TRUE;  
}
```

This is the .ass file that renders the above image:

```
options  
{  
    AA_samples 6  
    outputs "RGB RGB myfilter mydriver"  
    xres 640  
    yres 480  
    GI_diffuse_depth 1  
}  
  
driver_jpeg  
{  
    name mydriver  
    filename "randflake.jpg"  
    gamma 2.2  
}  
  
gaussian_filter  
{  
    name myfilter  
}  
  
plane  
{  
    name myplane  
    point 0 -10 0  
    normal 0 1 0  
    shader planeshader  
}  
  
procedural  
{  
    name myrandflake  
    dso "./libRandFlake.so"  
    min -10.05 -10.05 -10.05  
    max 10.05 10.05 10.05  
    shader flakeshader  
    declare G_npoints constant INT  
    declare G_recursions constant INT  
    declare G_sphererad constant FLOAT  
    declare G_cloaddrad constant FLOAT  
    declare G_seed constant INT  
    G_npoints 6  
    G_recursions 9  
    G_sphererad 0.015  
    G_cloaddrad 10  
    G_seed 4  
}
```

```
lambert
{
    name flakeshader
    Kd 0.7
}

lambert
{
    name planeshader
    Kd_color .15 .2 .2
}

persp_camera
{
    name mycamera
    focus_distance 11
    aperture_size .15
    position -5 5 15
    look_at 0 0 0
}

point_light
{
    name key
    position 100 200 100
    radius 4
    color 1 0.6 0.4
}

spot_light
{
    name kicker
    position -200 200 -500
    look_at 0 0 0
    cone_angle 2
    color .1 0.2 3
}

skydome_light
{
    name mysky
    color 0.6 0.85 1
```

```
    intensity .5  
}
```

The compilation commands:

```
c++ -o random_flake.os -c -fPIC -D_LINUX -fPIC -I. -I/opt/arnold/include random_flake.cpp  
c++ -o librandom_flake.so -shared random_flake.os -L/opt/arnold/bin -lai
```

## Shaders (API)

- Arnold Metadata Files
- Create a DLL that Contains Multiple Shaders
- Create an Empty VS 2008 Project to Compile a Shader into a DLL on Windows
- Create an Empty XCode 4 Project to Compile a Shader into a Dynamic Library on Mac OSX
- Creating a Shader Project with Xcode on Mac OSX
- Creating Refracted Rays With Proper Ray Derivatives
- Implementing MIS-sampled BRDFs
- Shaders Library
- Understanding GI With a Basic Shader
- Using Trace Sets
- Writing A Vector Displacement Shader
- Writing Your First Shader

## Arnold Metadata Files

Beginning with release 3.3.5, Arnold includes support for specification of nodes and attributes metadata using separate text files.

Each library implementing Arnold nodes could have an associated metadata file, with the same name of the library and the extension ".mtd". For example, a library called mtoa\_shaders.dll would have a corresponding mtoa\_shaders.mtd. This metadata files will be automatically loaded when the library is first loaded. As a special case, for built-in Arnold nodes, the name of the metadata file is arnold.mtd, as this library has different names on different platforms.

Also, there could be additional metadata files, which can be loaded on demand, by using the API function provided in ai\_metadata.h (or ai\_metadata.py for Python):

```
AI_API AtBoolean AiMetaDataLoadFile(const char* filename);
```

This files could be used for example to override values or add additional metadata to built-in nodes, such as MtoA and SItmA do for Arnold built-ins.

Finally, third party shader developers could include a metadata file with their library, in a file with the same name as the library, but ".mtd" extension. In this metadata file, they could include information for all plugins (SItmA, MtoA for now), so that the same file would work for all of them.

### File Format

The format of the metadata files is like this:

```
# Arnold test metaData file
[node wireframe]
desc STRING "This shader can be used to display the edge structure of a geometry."
[attr line_width]    min INT -3
                    max INT 100
                    softmin INT 0
                    softmax INT 1
                    desc STRING "Sets width of the lines used to display edges."
[attr fill_color]   desc STRING "Color used for the faces."
[attr line_color]   desc STRING "Color used for the lines used to display edges."
[attr raster_space] desc STRING "Uses screen space to measure line width."
[attr edge_type]    desc STRING "Selects base geometry element for edge display."
[attr example]      maya.test1 FLOAT -0.63
                    maya.test2 BOOL true
                    maya.test3 BOOL 0
                    maya.test4 RGB 0 1 0
                    maya.test5 POINT2 -1 489
                    maya.test6 INT 0x09ABCDEF
```

There could be any number of nodes in the file. There can also be multiple node sections for the same node (metadata will be applied in the order they are found in the file).

As a convention, we are using metadata names starting with "maya." for Maya specific metadata, and "xsi." for XSI specific metadata. There are some other conventions for standard metadata elements that would be published on a separate document.

### Maya Node IDs

Maya needs a unique ID assigned to nodes to keep track of custom nodes.

Just add the "maya.id" integer metadata. hex formatting is supported in .mtd file as well.

## Create a DLL that Contains Multiple Shaders

### Multiple shaders in a single DLL/DSO

You can have as many shaders as you want in a DLL (or .so shared library for you Linux folks). When Arnold loads the DLL, it will call the **node\_loader** entry point to get the list of available shaders.

Each shader should be defined in its own source code file, which then has to export its methods with the following macro:

```
AI_SHADER_NODE_EXPORT_METHODS(MyShaderMethods);
```

The **MyShaderMethods** name can be replaced by anything else, as long as it is unique for this DLL.

These methods are then imported from the file where **node\_loader** is defined:

```
extern AtNodeMethods* MyShaderMethods;
extern AtNodeMethods*
MyOtherShaderMethods;
```

The **node\_loader** function has the following signature:

```
bool node_loader(int i, AtNodeLib
*node)
```

When Arnold loads the DLL, it will call this entry point several times, each time increasing the index **i**, until one of the calls returns FALSE. So, for example, if you have two shaders, your **node\_loader** function should return TRUE for values of **i** between 0 and 1 (inclusive), and FALSE for all other values. Then, for values in the [0..1] range, it should set the corresponding shader data in the **AtNodeLib** struct.

The easiest way to do this is using a switch statement like this:

```
enum SHADERS
{
    MY_SHADER,
    MY_OTHER_SHADER
};

node_loader
{
    switch (i)
    {
        case MY_SHADER:
            node->methods      = (AtNodeMethods*) MyShaderMethods;
            node->output_type  = AI_TYPE_RGB;
            node->name         = "MyShader";
            node->node_type    = AI_NODE_SHADER;
            break;

        case MY_OTHER_SHADER:
            node->methods      = (AtNodeMethods*) MyOtherShaderMethods;
            node->output_type  = AI_TYPE_RGBA;
            node->name         = "MyOtherShader";
            node->node_type    = AI_NODE_SHADER;
            break;

        default:
            return false;
    }

    sprintf(node->version, AI_VERSION);
    return true;
}
```



## Create an Empty VS 2008 Project to Compile a Shader into a DLL on Windows

### Prerequisites

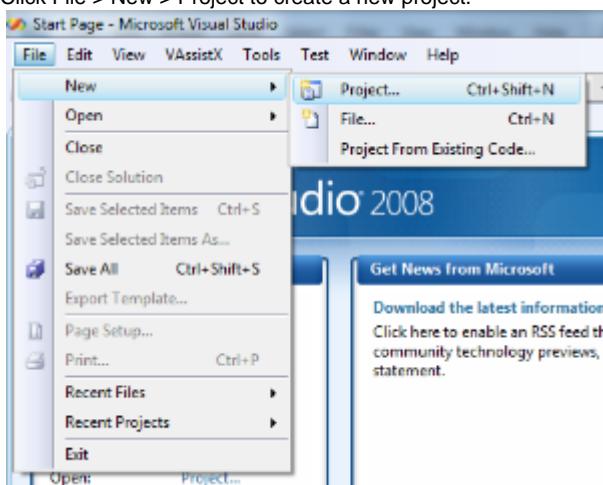
- Visual Studio 2008 Express
- Windows SDK (.NET Framework 3.5 sp1)

This link may be useful for 64-bit compilation and the express edition: <http://jenshuebel.wordpress.com/2009/02/12/visual-c-2008-express-edition-and-64-bit-targets/>

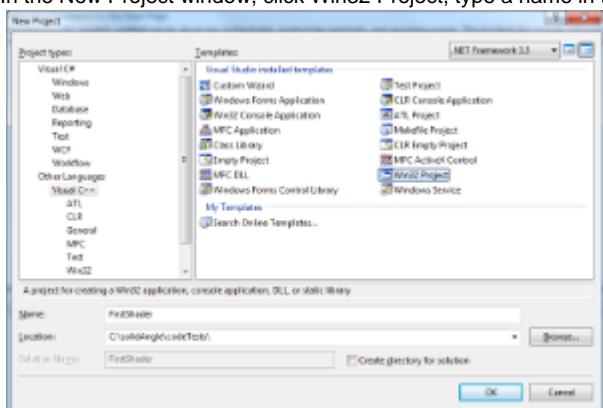
### Creating an empty shader project

Here are the steps to create a project in Visual Studio 2008 and compile a shader into a DLL. If you are using another compiler, go to the bottom of the tutorial.

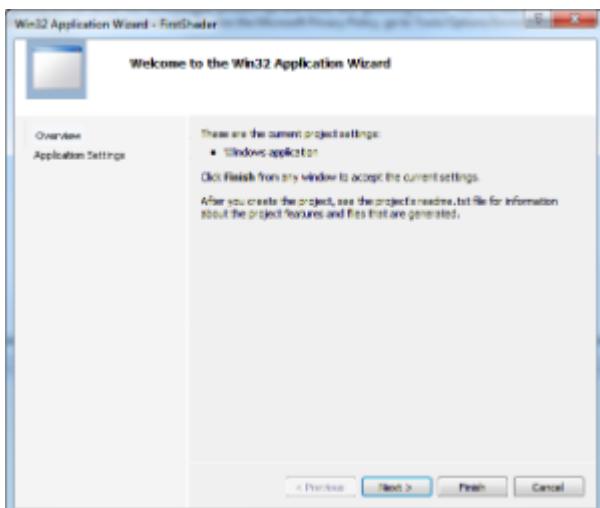
1. Click File > New > Project to create a new project.



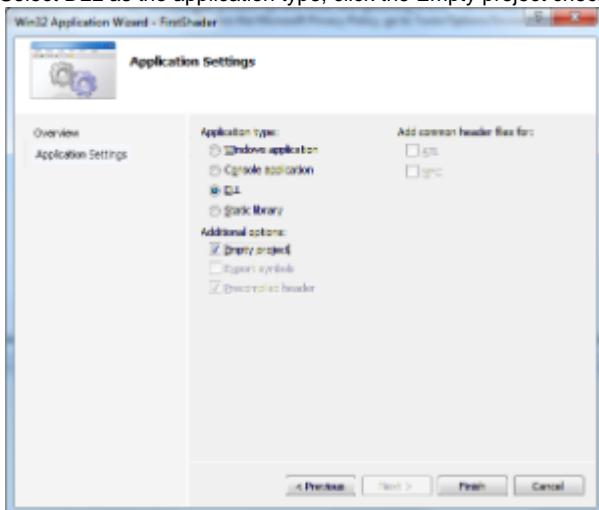
2. In the New Project window, click Win32 Project, type a name in the Name box, and click OK.



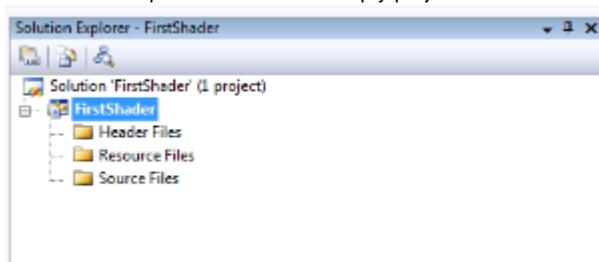
3. In the Win32 Application Wizard, click Next.



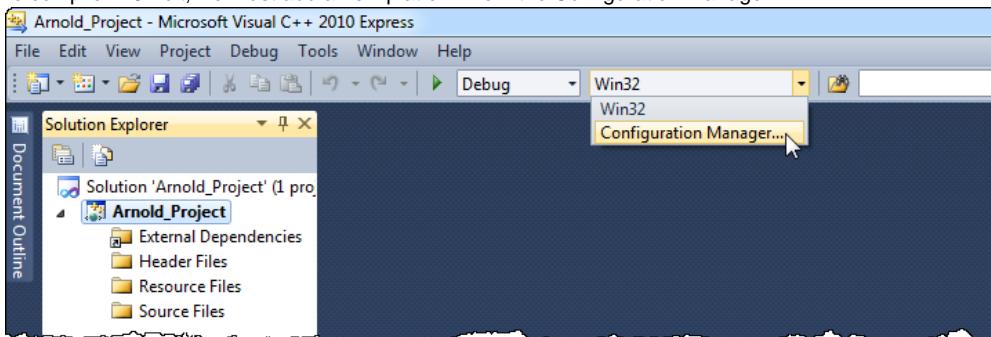
4. Select DLL as the application type, click the Empty project checkbox, and click Finish.



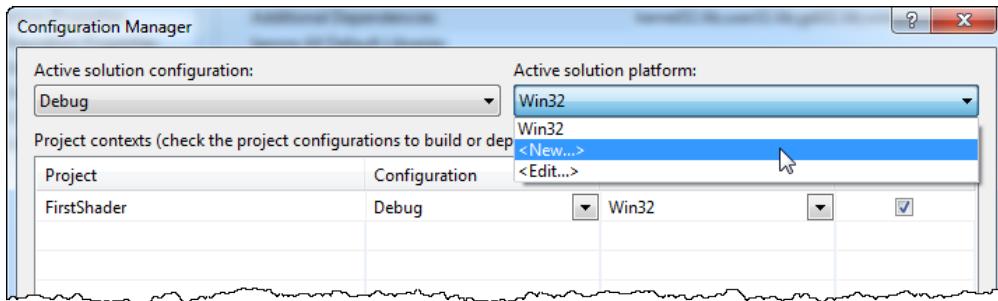
The Solution Explorer will show an empty project:



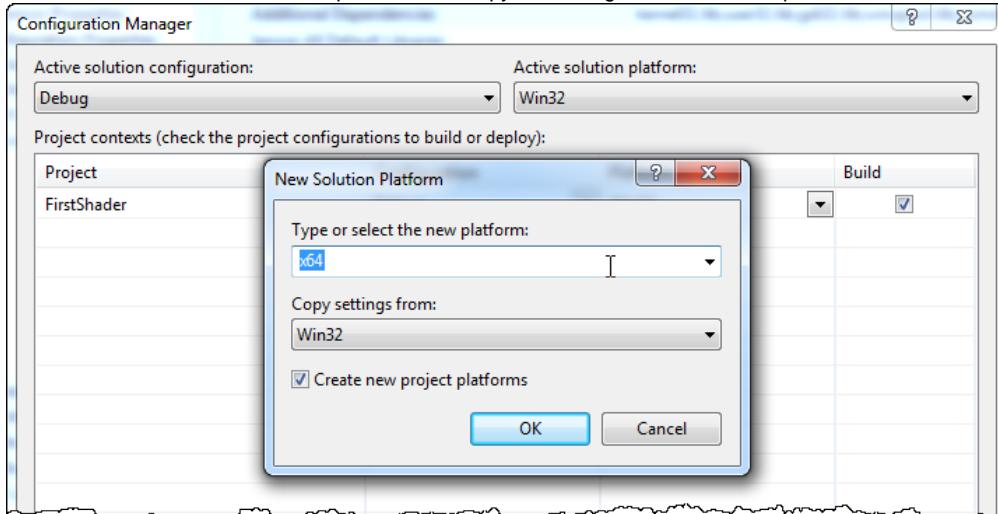
5. To compile in 64-bit, we must add a new platform from the Configuration Manager:



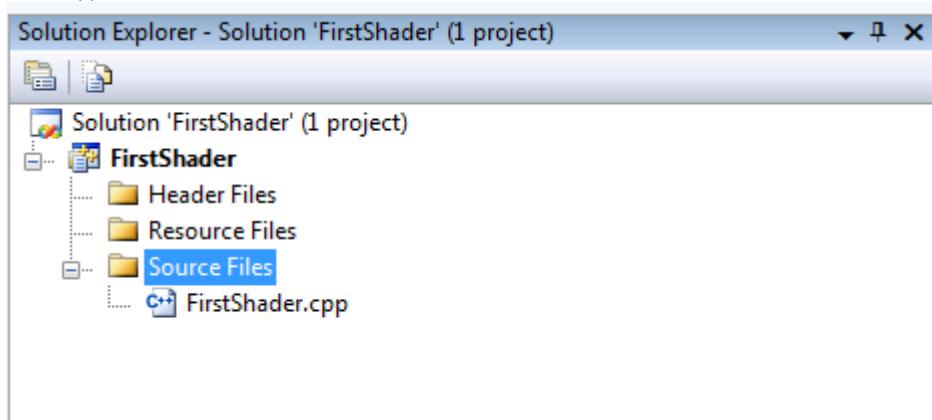
6. In the Active solution platform list, click <New...>.



And then select x64 as the solution platform, and copy the settings from the Win32 platform.

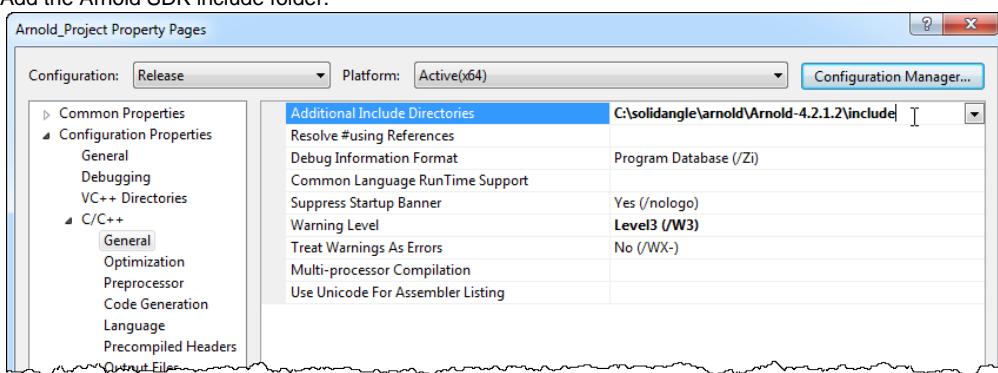


7. Add a .cpp file or create one:

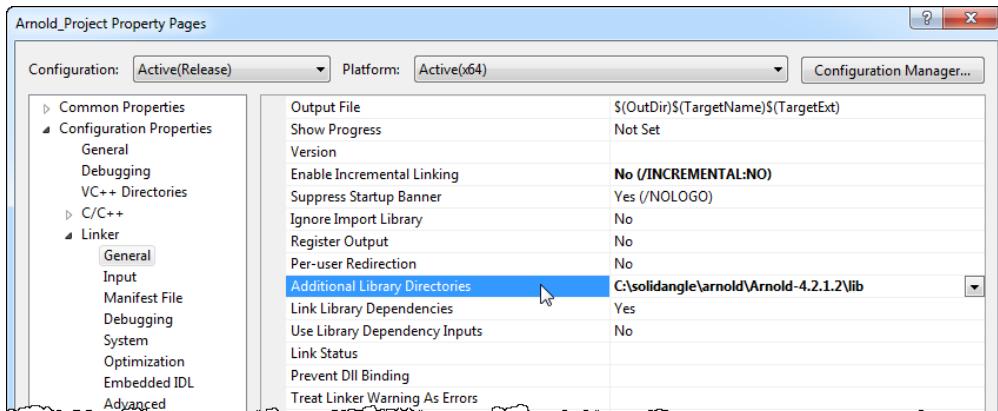


8. We now have to configure the project. Right-click the project and then click Properties.

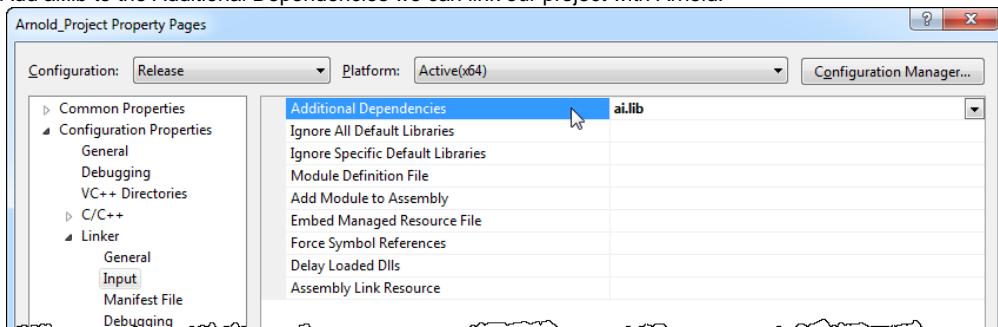
9. Add the Arnold SDK include folder.



10. Then we add the Arnold lib folder where the ai.lib file is located:



11. Add ai.lib to the Additional Dependencies we can link our project with Arnold:



You should now be able to compile the project. This will create a DLL in a folder next to the Solution (depending on the configuration). You must copy this .dll file into your target Arnold project. If you are using Softimage, you'll want to copy the file to the SitoA bin folder, for example C:\solidangle\sitoa\Addons\SitoA\Application\Plugins\bin\nt-x86-64.

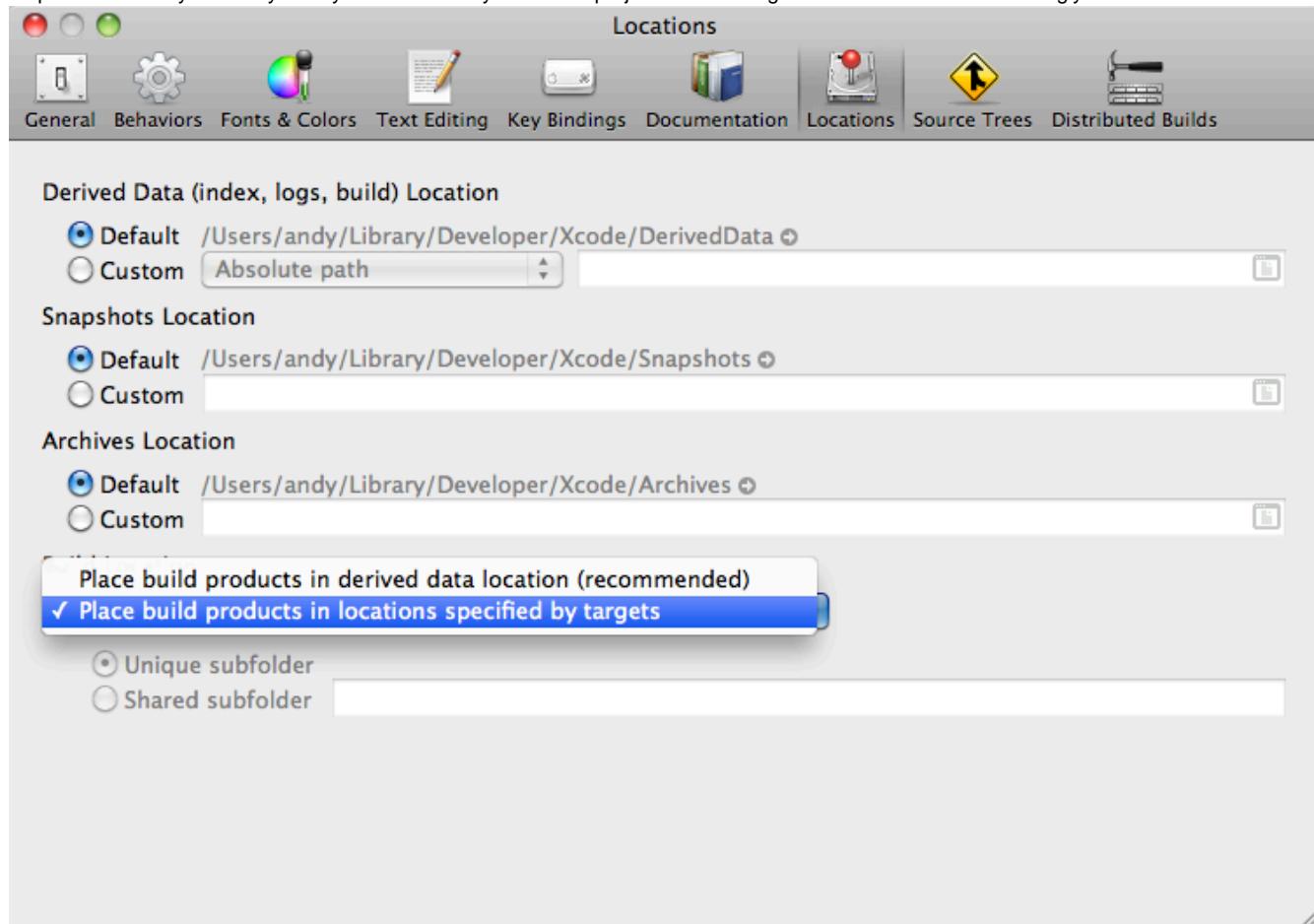
## Other Compilers

If you are trying to use Visual Studio Express make sure to follow these additional steps:

1. First download the Windows SDK 7.1 from the Microsoft website (you MUST have the SDK installed FIRST before trying to switch compiler configurations). The link to the website is [here](http://www.microsoft.com/en-us/download/details.aspx?id=4422).
2. Verify that you have the x64 compiler for Windows by going into Start->Control Panel->Programs and Features. Scroll down to the bottom and you should see this: Microsoft Visual C++ 2010 Compilers Standard - enu - x64. If you do NOT see this, then you didn't install the SDK properly! If you already installed the Visual C++ 2010 before the SDK you should deinstall the 2010 x86/x64 redistributables otherwise the SDK will not install. Then install the SDK but without compilers. After that you should install the 2010 compiler update : <http://www.microsoft.com/en-us/download/details.aspx?id=4422> Also make sure that you installed the 2010 C++ Sp1 before : <http://www.microsoft.com/en-us/download/details.aspx?id=23691>
3. You must download the Visual C++ Express package (2008 or 2010). You can get this download here [2008](#) or [2010](#).
4. Follow the instructions for compiling to 64-bit from the tutorial
5. IMPORTANT: At this time, it will NOT link. It will complain about not finding the file kernel32.lib. This is because Visual Studio is pointing to the Win32 library directories instead of the x64 directories. You should again point Visual Studio to the new location: C:\Program Files\Microsoft SDKs\Windows\v7.1\Lib\x64 in the Linker->General->Additional Directories tab.

## Create an Empty XCode 4 Project to Compile a Shader into a Dynamic Library on MacOSX

Here are the steps needed to create a project in Xcode 4 on Mac OSX and compile a shader into a shared library. Launch Xcode and go to the preferences. If you want your .dylib to build into your Xcode project folder change the 'Build Location' accordingly.



- Choose a Mac OSX Empty Project (under 'Other'):

**Choose a template for your new project:**

The screenshot shows the 'Choose a template for your new project' dialog. On the left, there are two columns of templates:

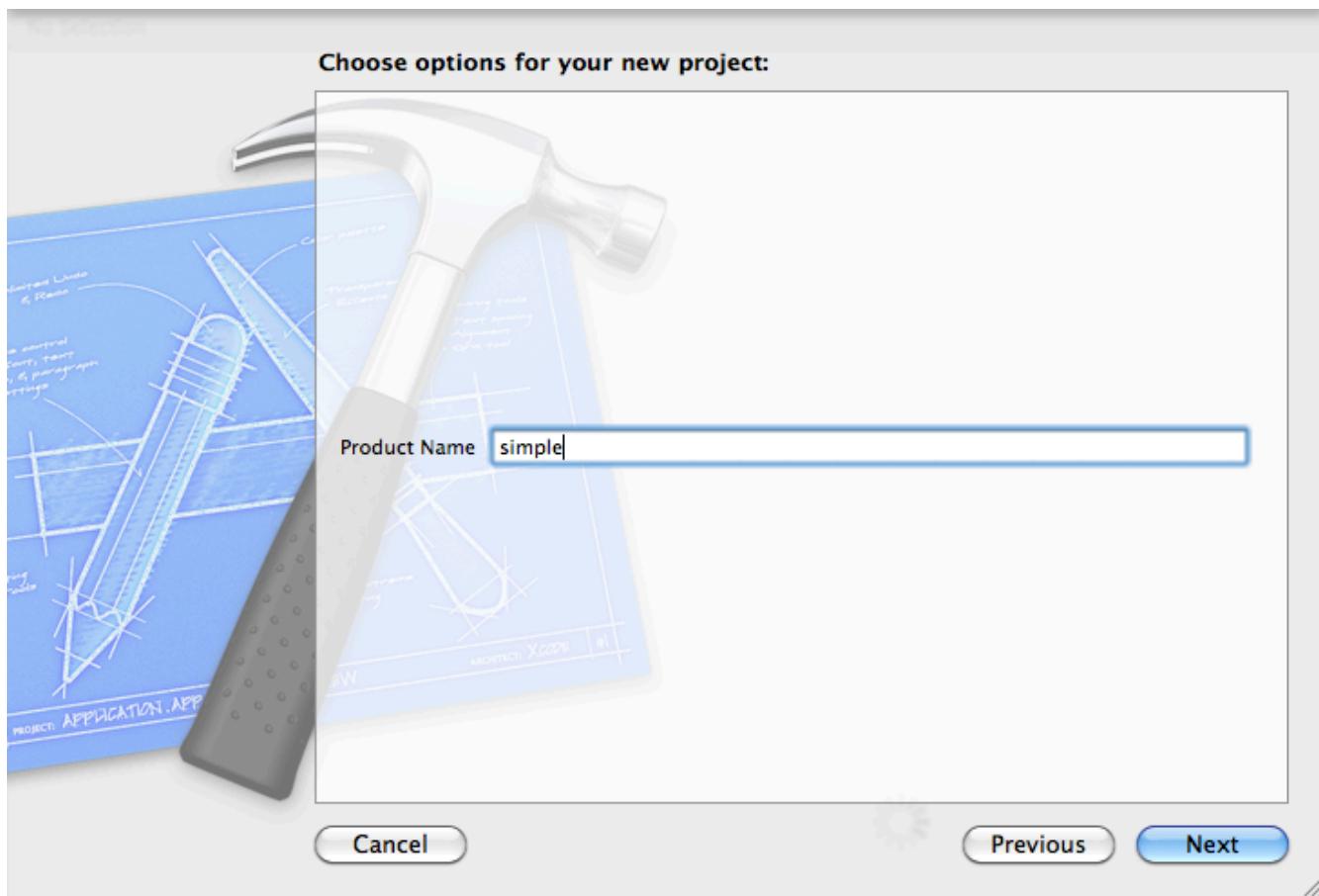
- iOS**: Application, Framework & Library, Other.
- Mac OS X**: Application, Framework & Library, Application Plug-in, System Plug-in, Other.

In the center, there is a preview area showing a blue folder icon labeled 'Empty'. To the right of the preview, the text 'External Build System' is displayed. At the bottom of the preview area, another 'Empty' folder icon is shown.

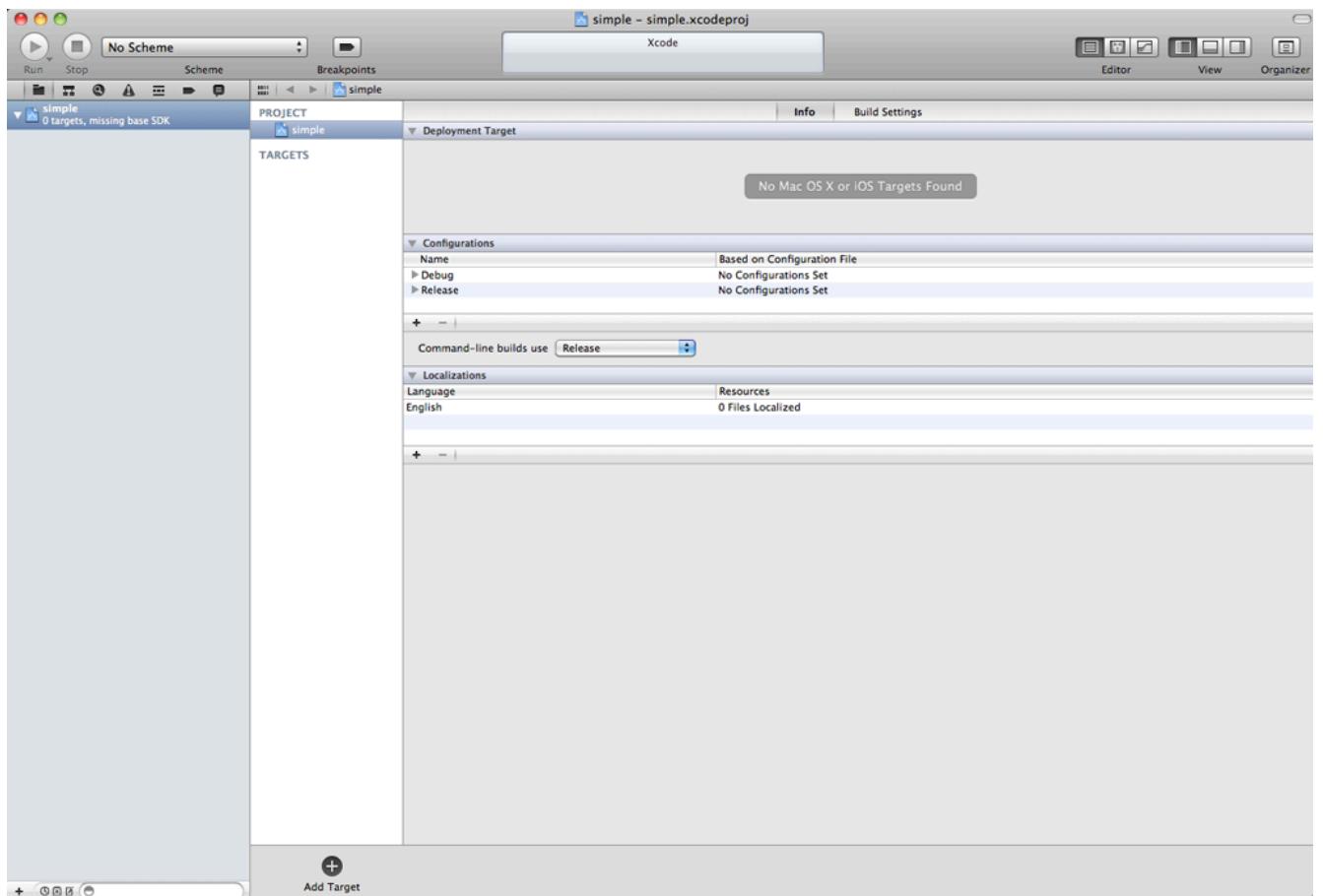
**Note:** This is an empty project with no files, targets, or build configurations.

At the bottom of the dialog are three buttons: 'Cancel', 'Previous', and 'Next'.

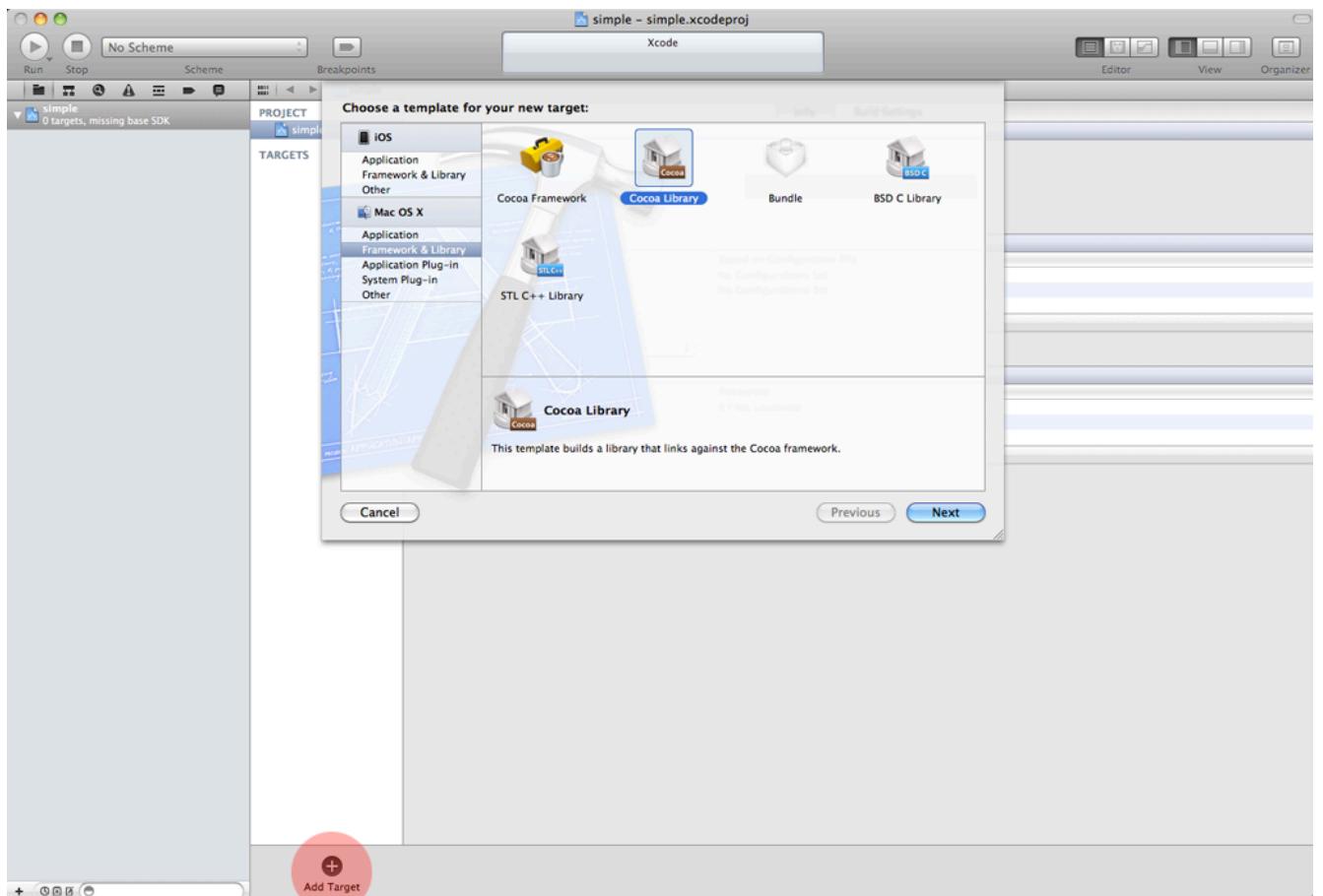
- Enter a project name, "simple" in this case.



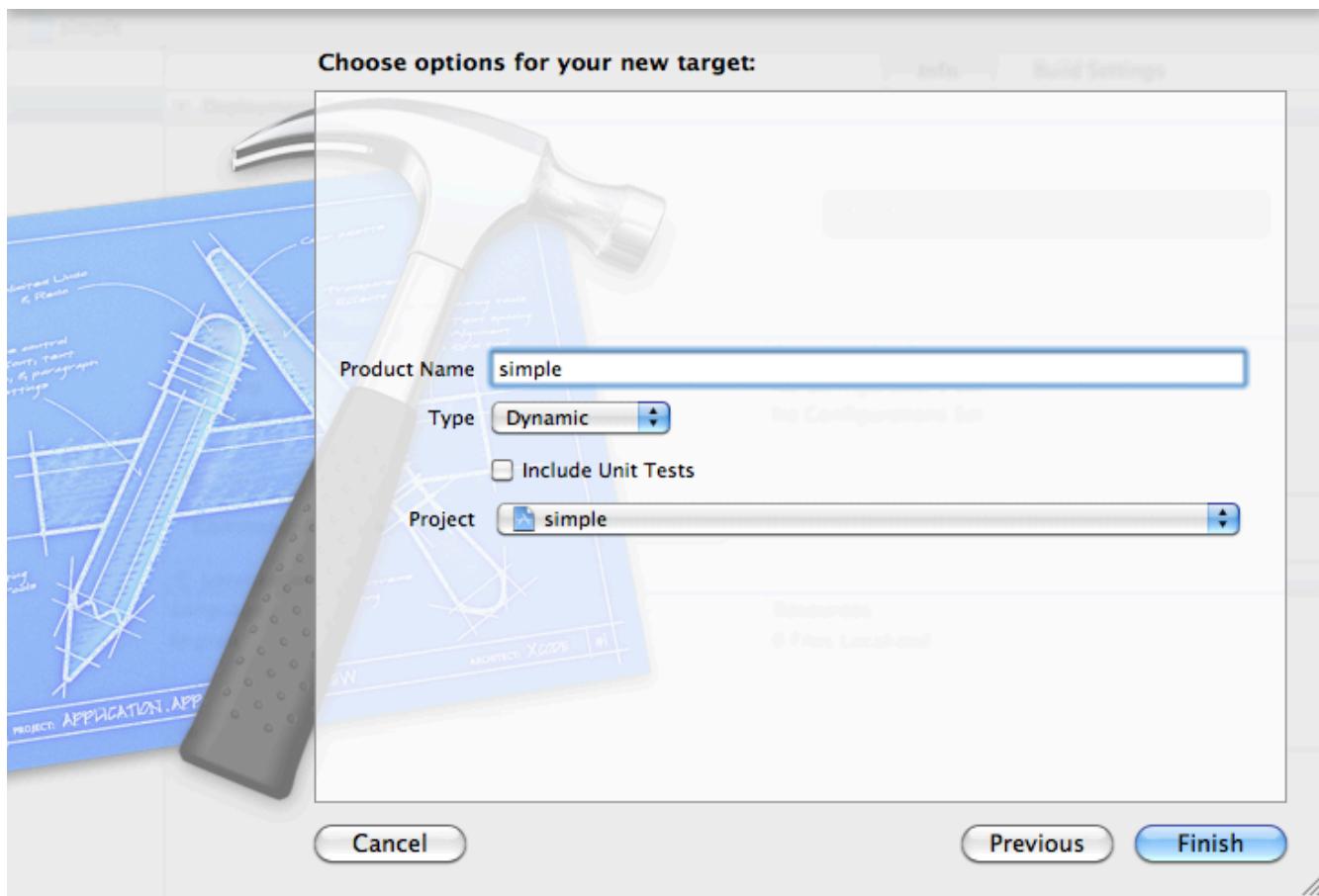
- Now you have a newly created empty project:



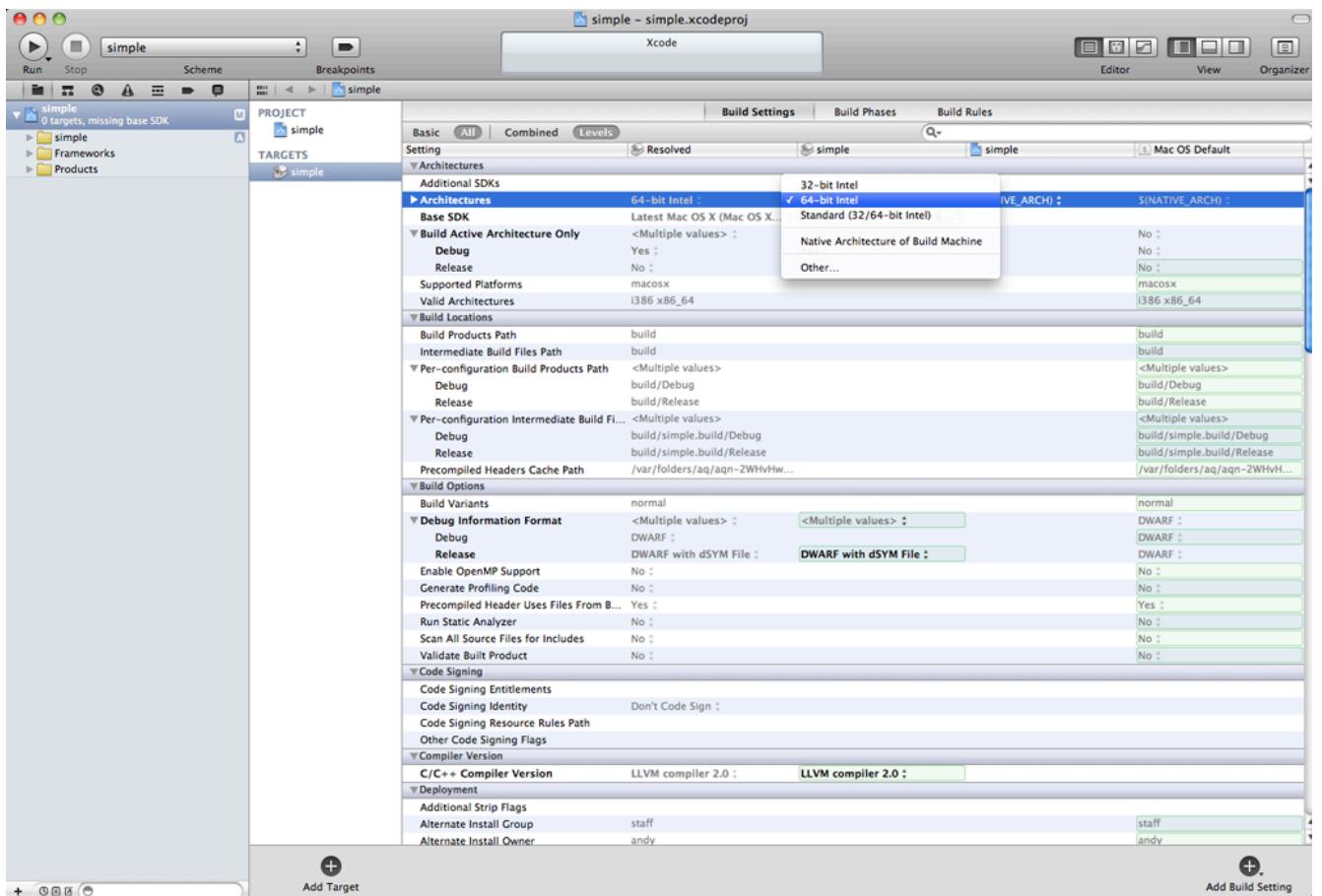
- Now let's add a target. This would be a Cocoa Library. Contrary to the Xcode 3 wiki documentation it's preferable to set up the target prior to creating new files. Doing so adds newly created files automatically to the target. If you choose to do it the other way around be sure to add your files to the target yourself.



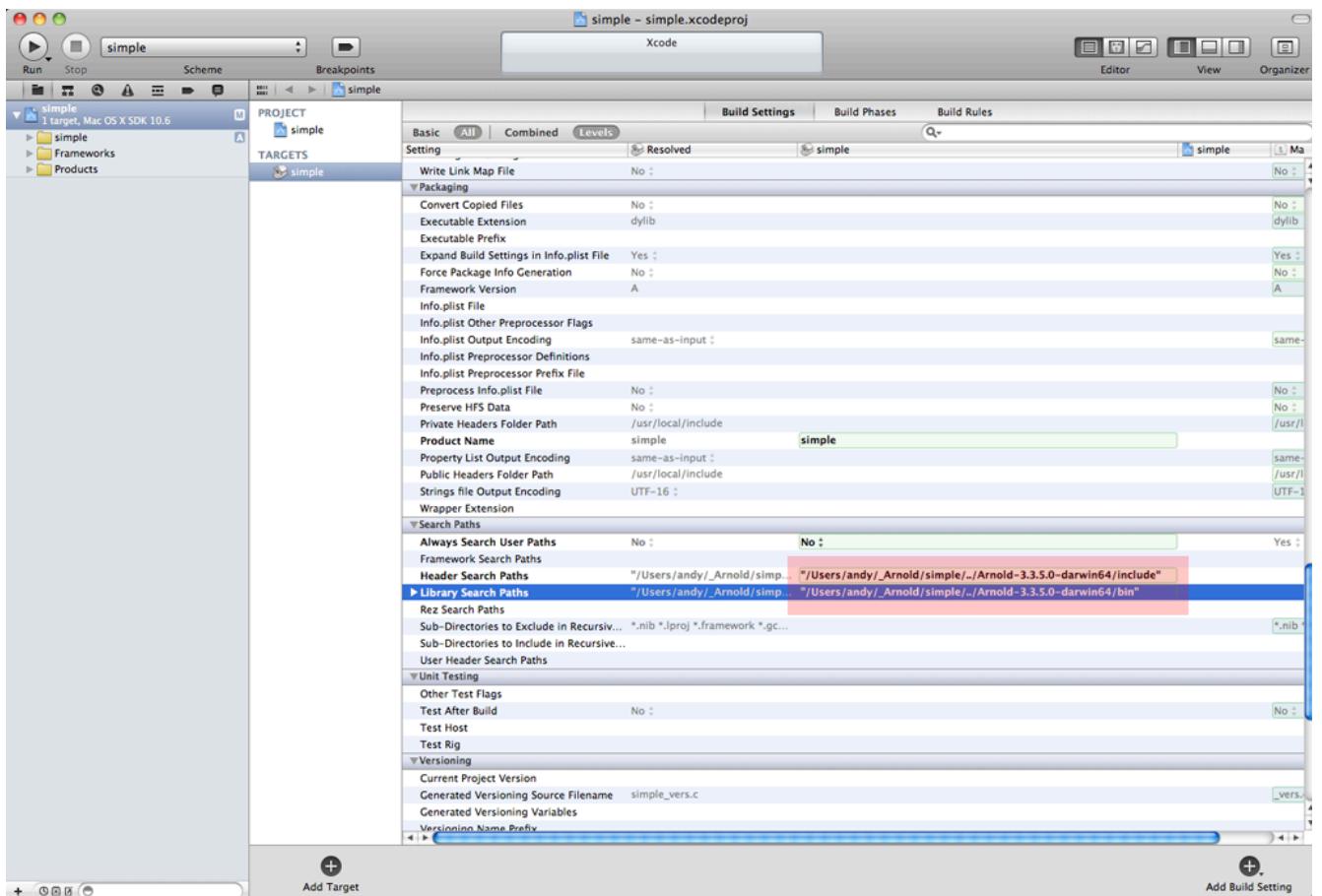
- Enter a name for your target. This is the name your file will end up with e.g. simple.dylib



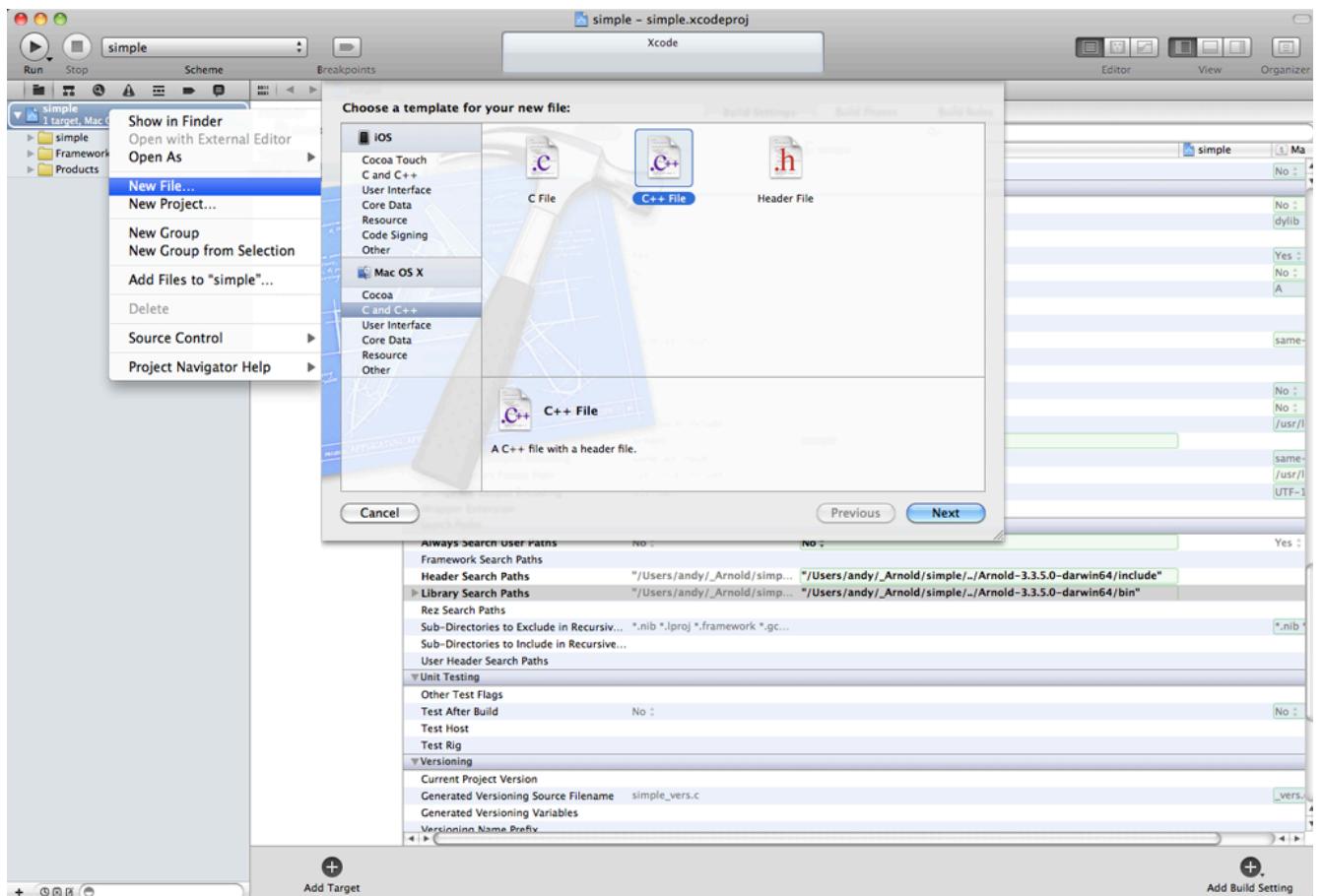
- It's important to set the architecture according to your Arnold distribution you're compiling against. If you don't set this explicitly, you would probably get link errors.



- Now set up the search paths for Arnold headers and libraries:



- You're now ready to add the shader source file and name it "simple.cpp"

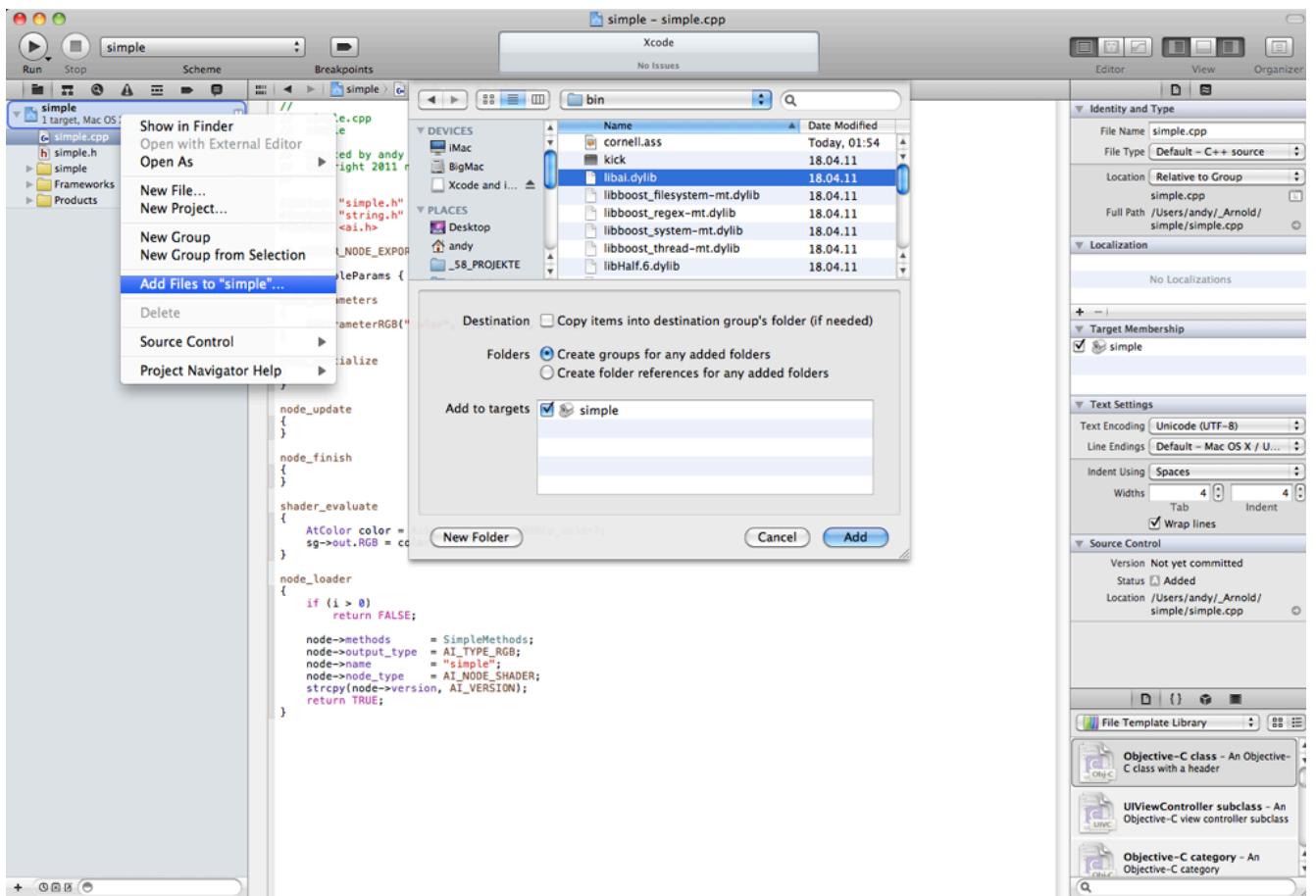


- Geek out on the shader code. Notice in the inspector to the right that the target is already set.

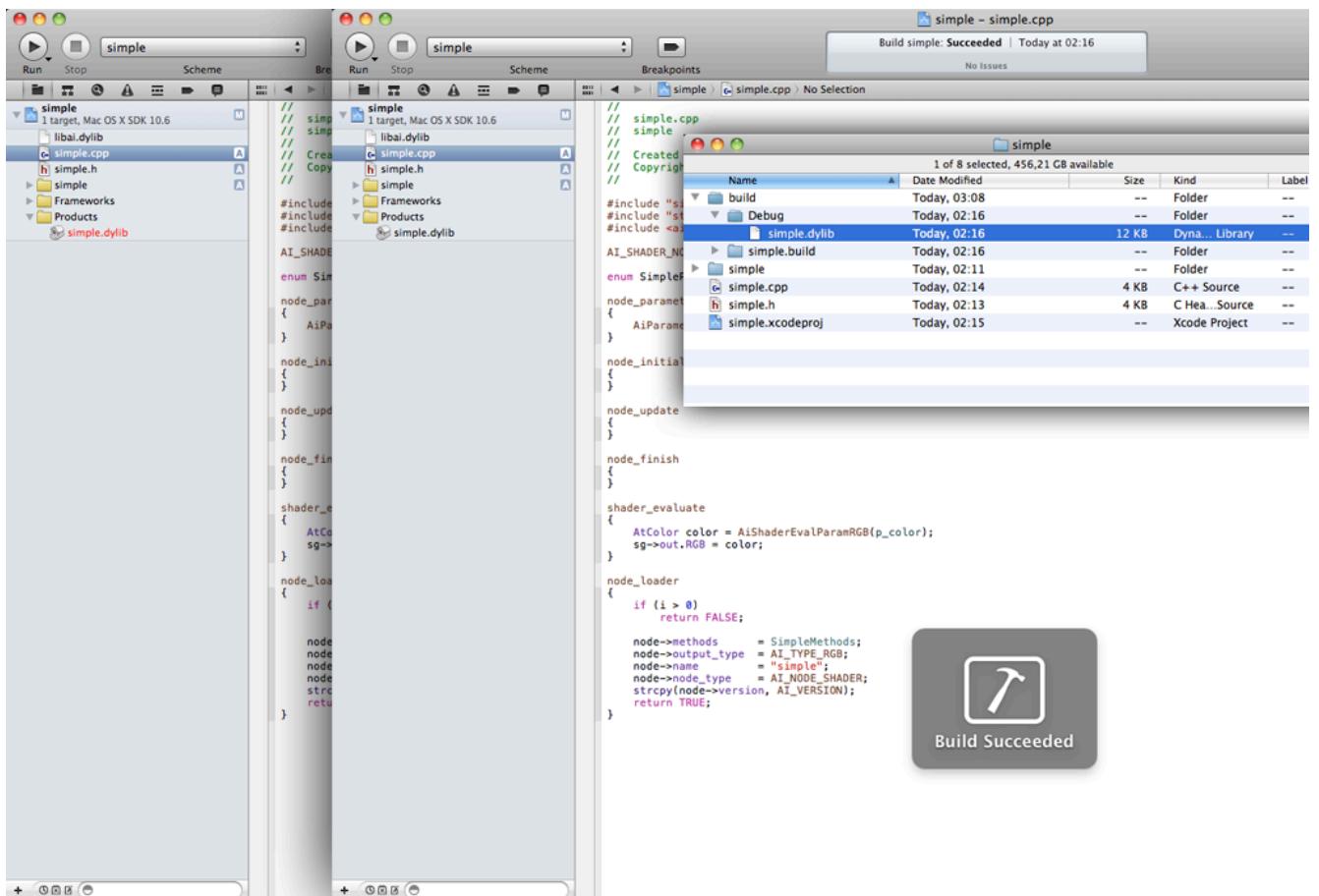
The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows a project named "simple" with files "simple.cpp" and "simple.h".
- Editor:** Displays the content of "simple.cpp". The code is a C++ shader implementation for Arnold. It includes comments at the top, defines an enum "SimpleParams", and implements methods like node\_initialize, node\_update, node\_finish, shader\_evaluate, and node\_loader.
- Identity & Type Inspector:** Shows the file's properties:
  - File Name:** simple.cpp
  - File Type:** Default - C++ source
  - Location:** Relative to Group simple.cpp
  - Full Path:** /Users/andy/\_Arnold/simple/simple.cpp
- Text Settings:** Includes Text Encoding (Unicode (UTF-8)), Line Endings (Default - Mac OS X / Unix), Indent Using (Spaces), and Widths (Tab 4, Indent 4).
- Source Control:** Shows the file is Not yet committed, Added, and located at /Users/andy/\_Arnold/simple/simple.cpp.
- File Template Library:** Shows categories: Objective-C class, UIViewController subclass, and Objective-C category.

- Add the libai.dylib from the bin directory of your Arnold distribution you're compiling against.



- Click on the arrow next to the yellow products folder icon. Until now a red 'simple.dylib' should show. This means your dylib has not been created. Go to Menu->Product->Build and you'll end up with a black "simple.dylib" and the according file inside your project folder:



- Once this dynamic library is placed where Arnold can find it you should be able to use it (for example place it in the bin directory under Arnold then type "kick -nodes" in the command-line).

Terminal 1: bash

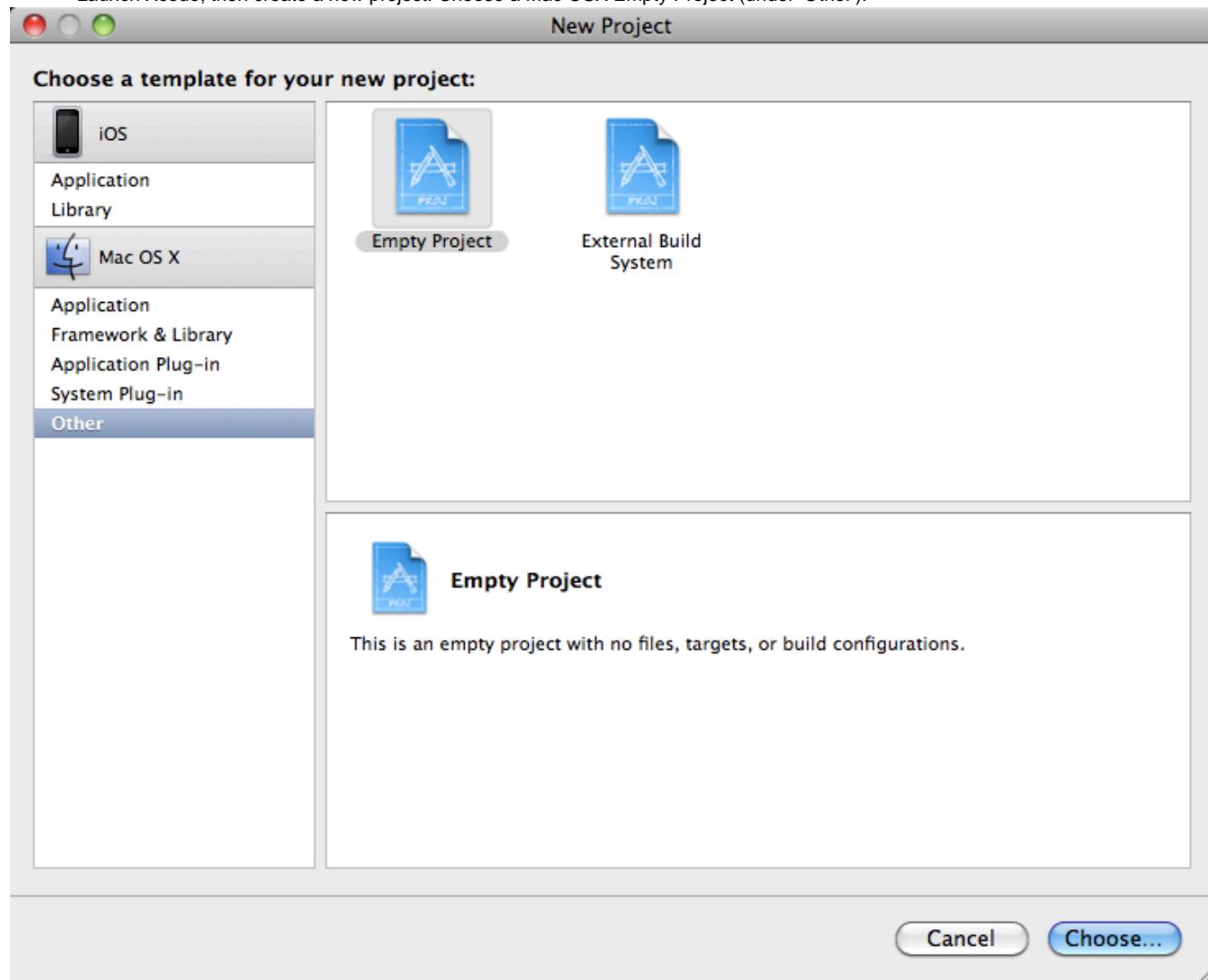
```
ginstance          shape
gobo               shader
hair               shader
heatmap_filter    filter
image              shader
implicit          shape
lambert           shader
light_blocker     shader
light_decay       shader
mtnet_filter      filter
motion_vector     shader
noise              shader
nurbs              shape
options            options
ortho_camera       camera
override           override
persp_camera       camera
plane              shape
point_light       light
points             shape
polymesh           shape
procedural         shape
quad_light         light
ray_switch         shader
sinc_filter        filter
sky                shader
skydome_light     light
sphere             shape
spot_light         light
standard           shader
triangle_filter   filter
utility            shader
variance_filter   filter
video_filter       filter
volume_scattering shader
wireframe          shader

loading plugins from .
simple             shader
iMac:bin andy$
```

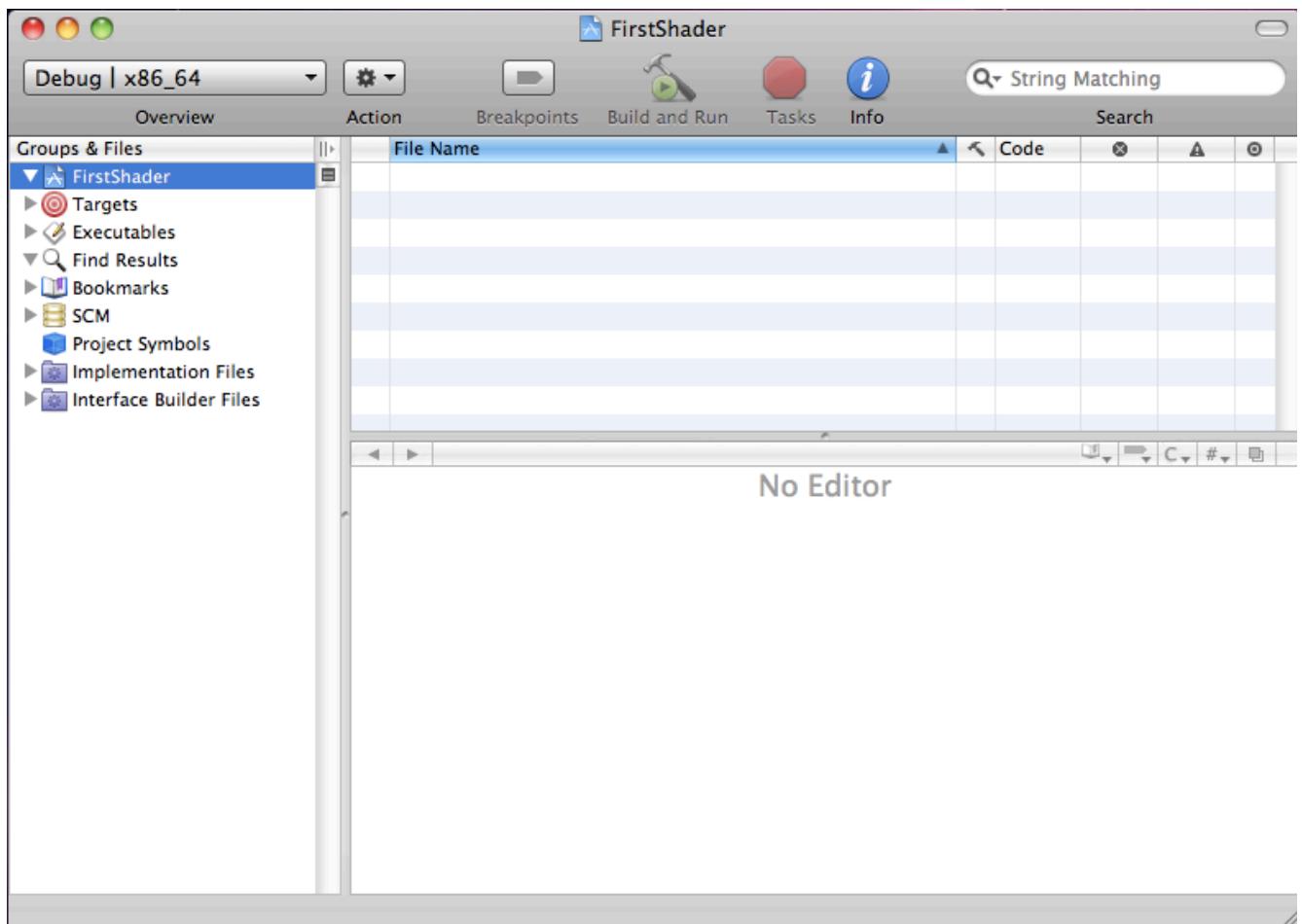
## Creating a Shader Project with Xcode on Mac OSX

Here are the steps needed to create a project in Xcode 3 on Mac OSX and compile a shader into a shared library.

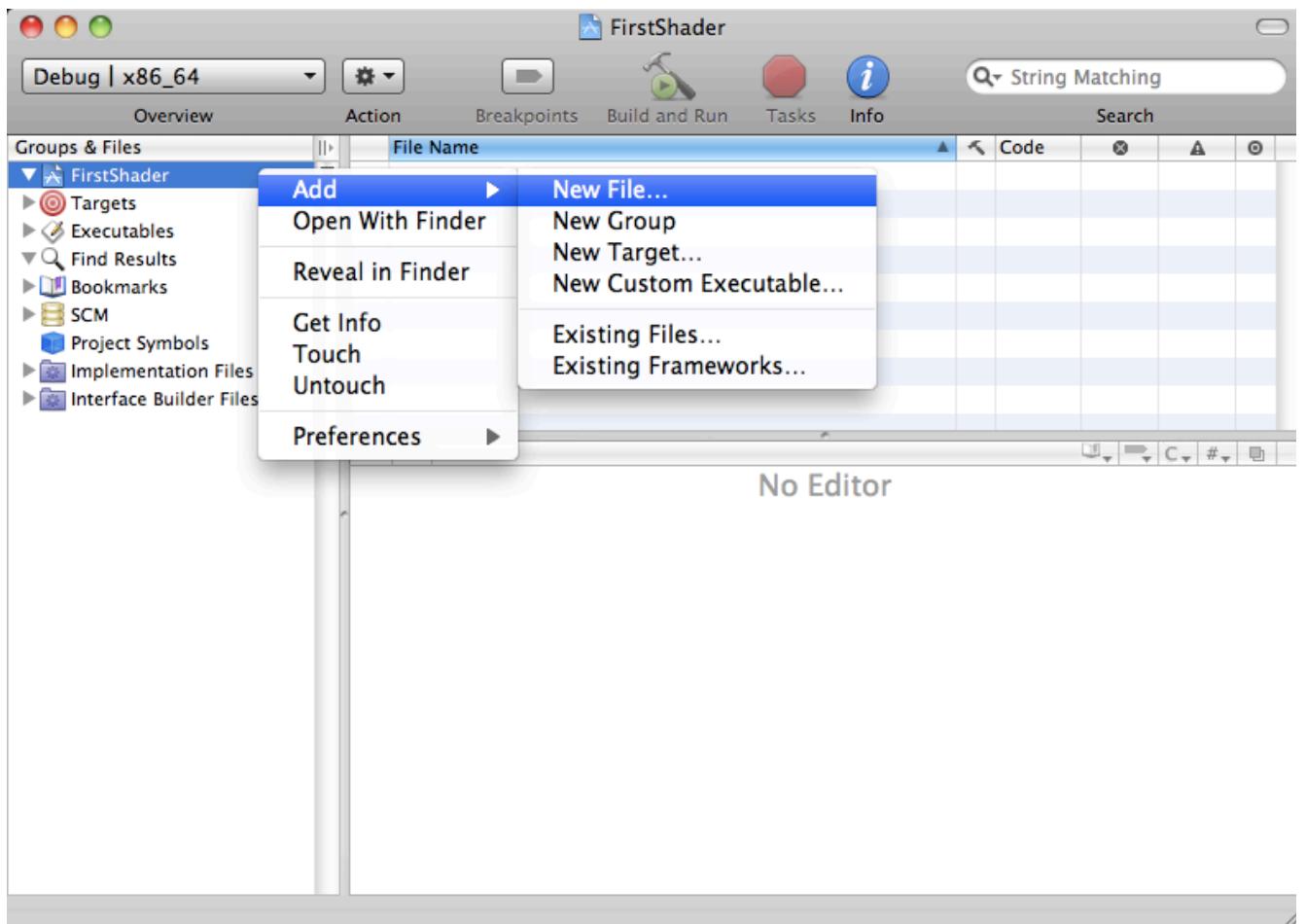
- Launch Xcode, then create a new project. Choose a Mac OSX Empty Project (under 'Other'):



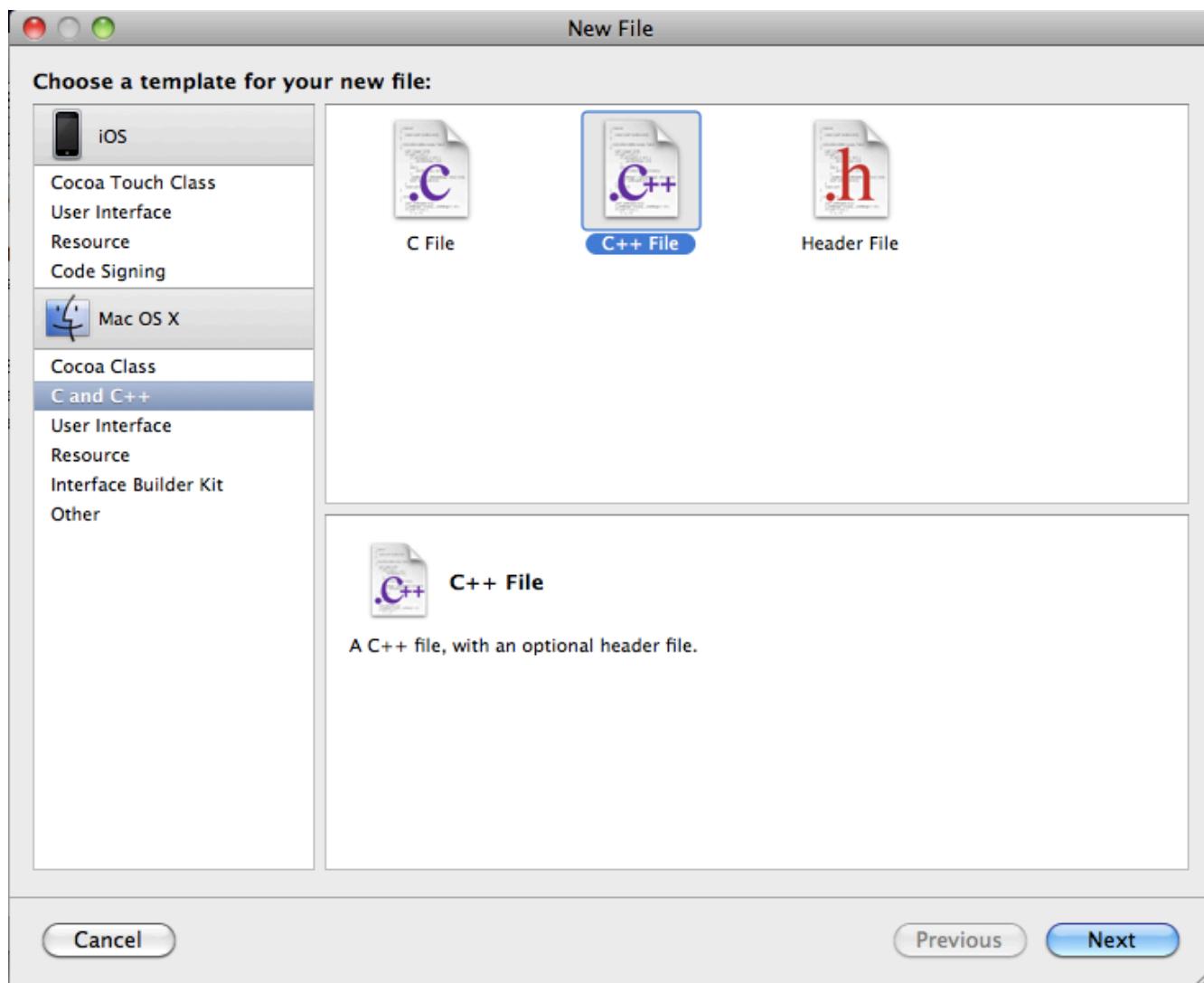
- When prompted, choose a name and location for the new project (for example a new folder called FirstShader, project name FirstShader). Xcode will then show your new empty project:



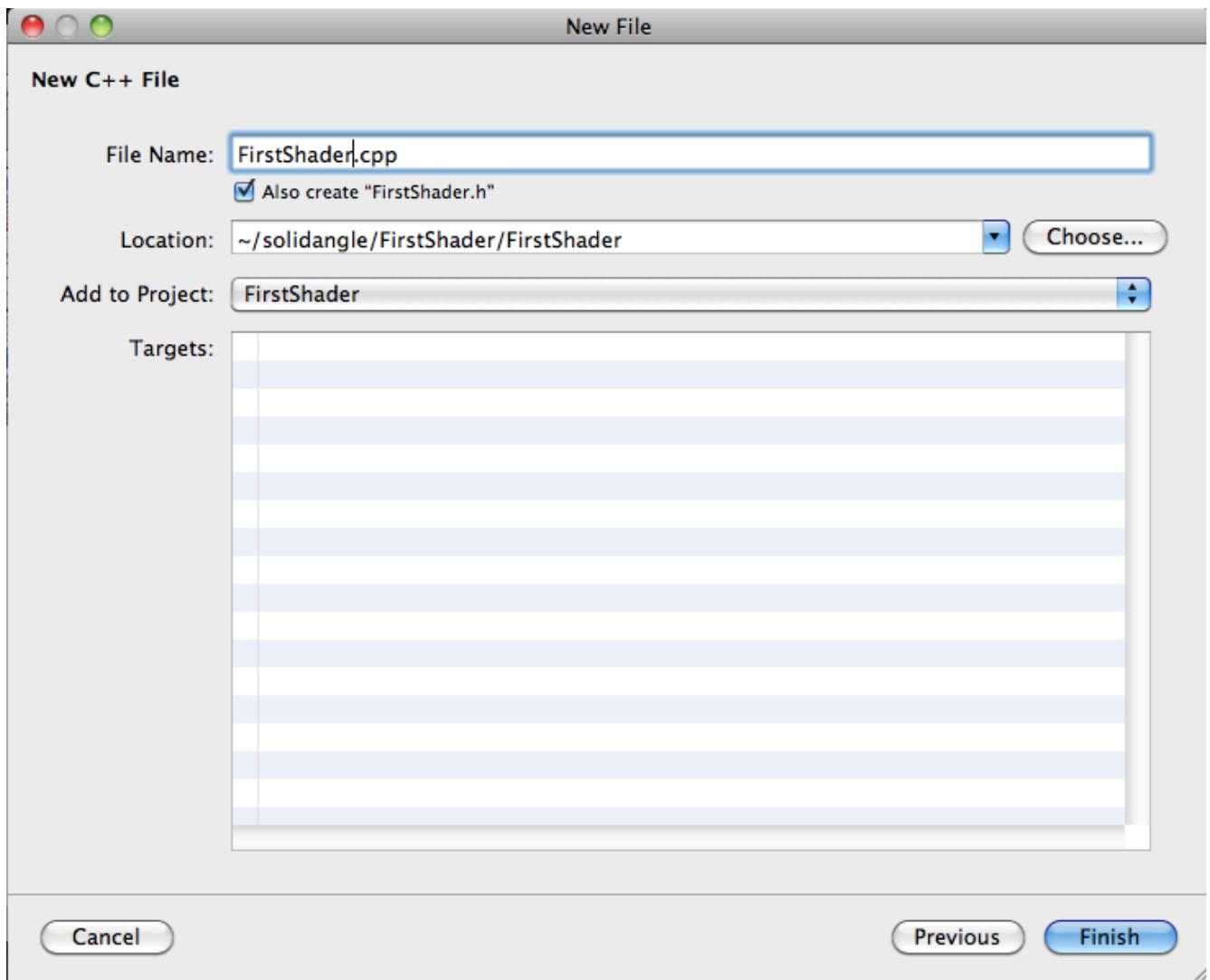
- Add a new C++ file by right clicking on the top level of the project and choosing 'New File' (or choose an existing file with 'Existing Files...'):



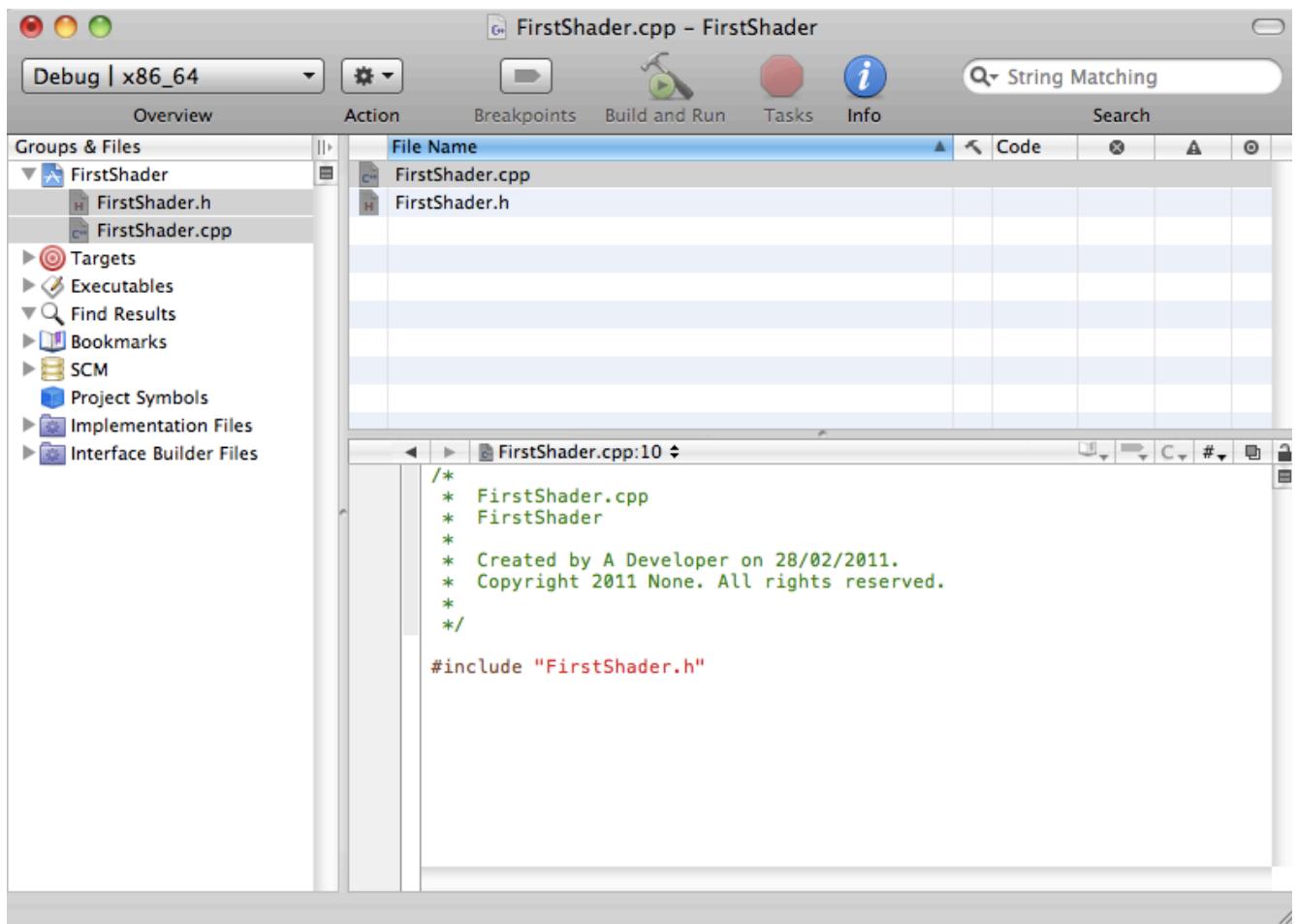
- You will need to specify that the new file is a C++ file:



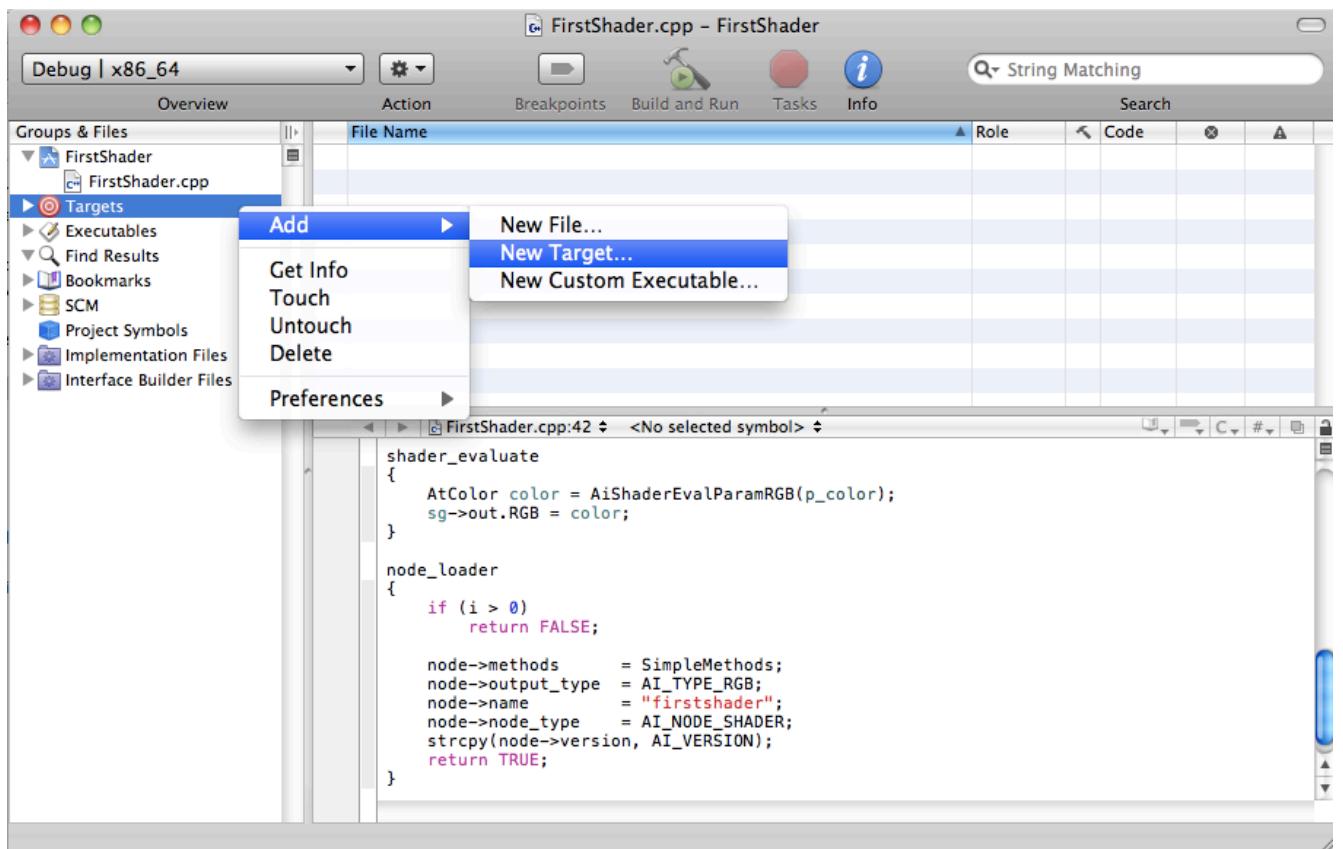
- Choose a location and name for the new file:



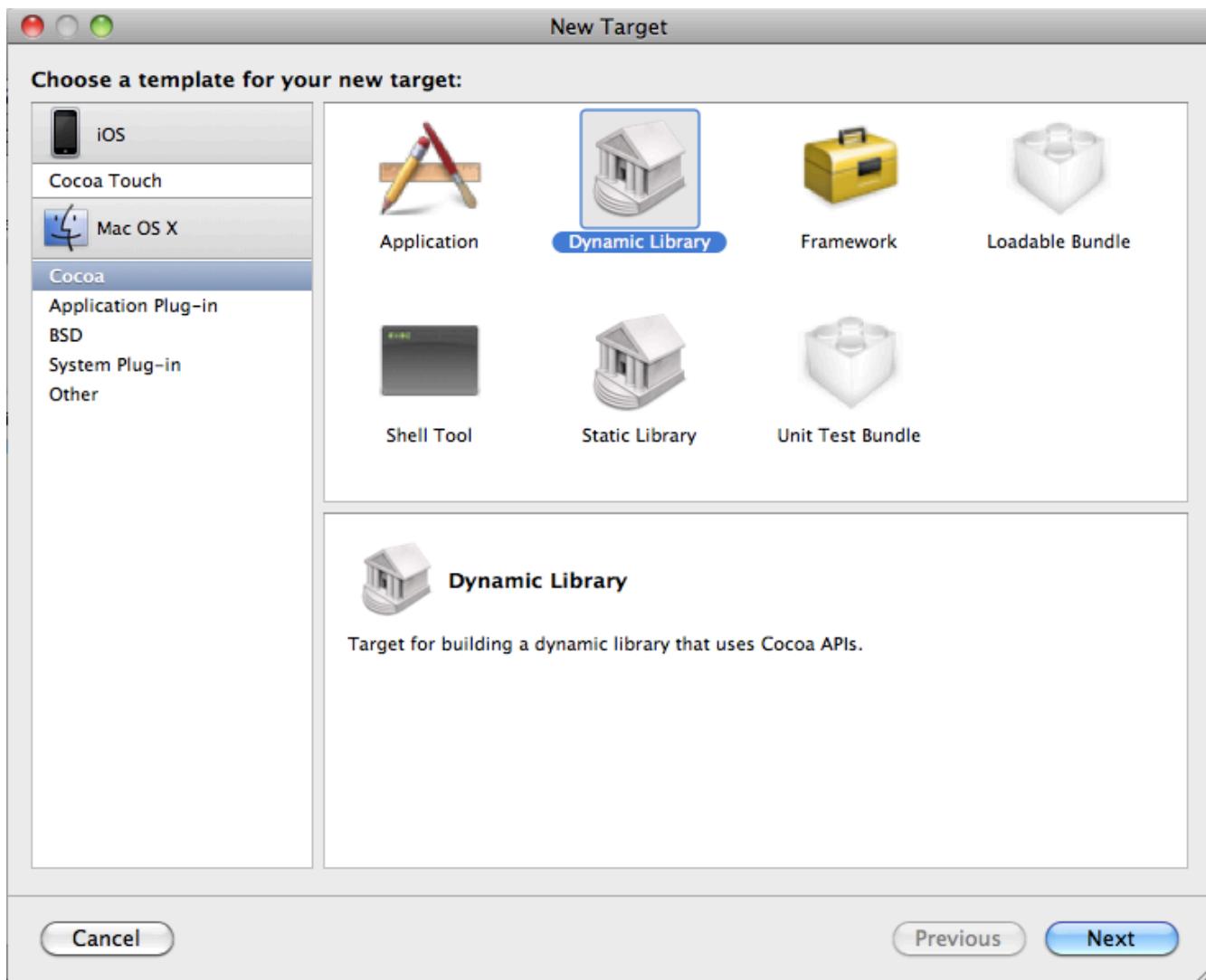
You should now have something like the following:



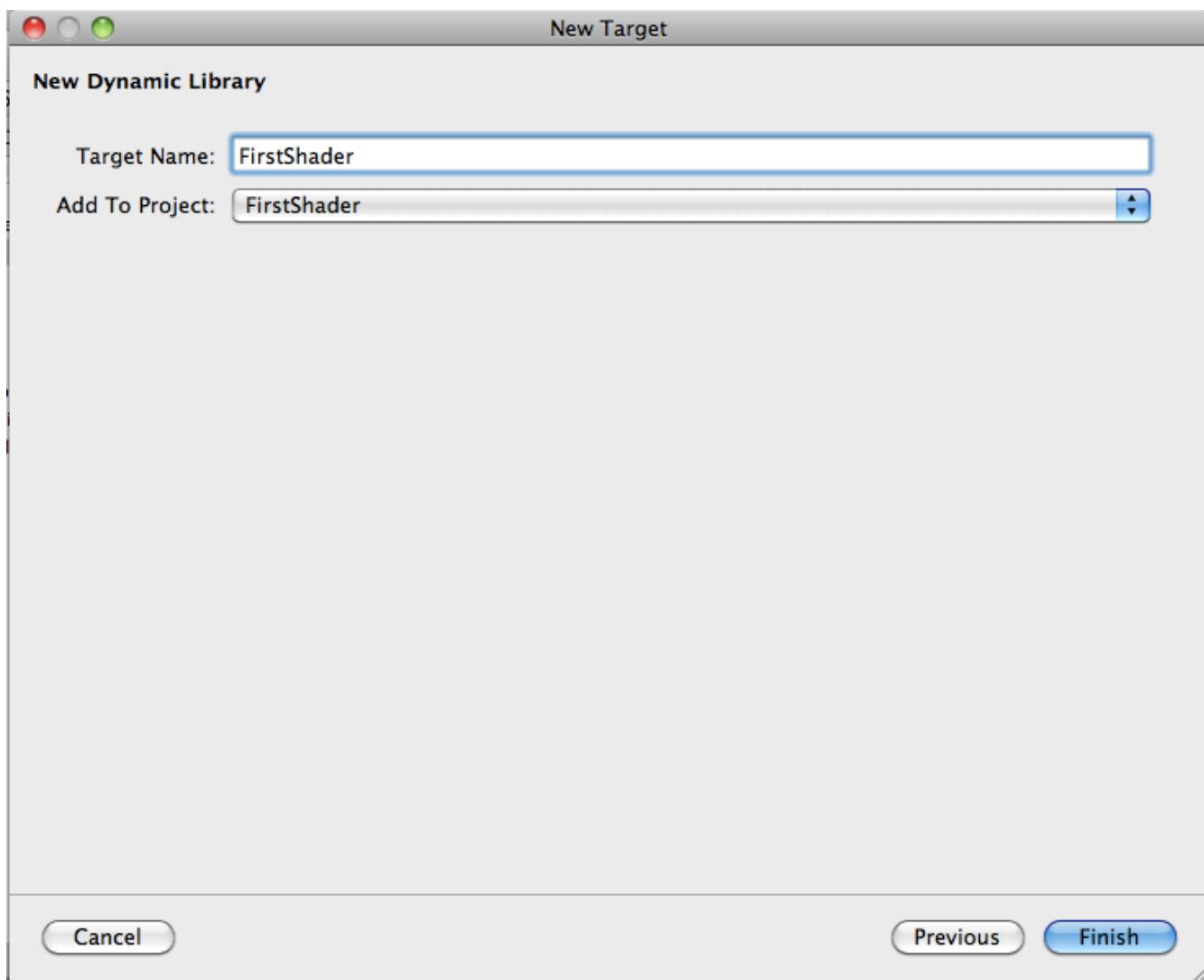
- Go ahead and add your code, but before building you will need to specify a build target for your project:



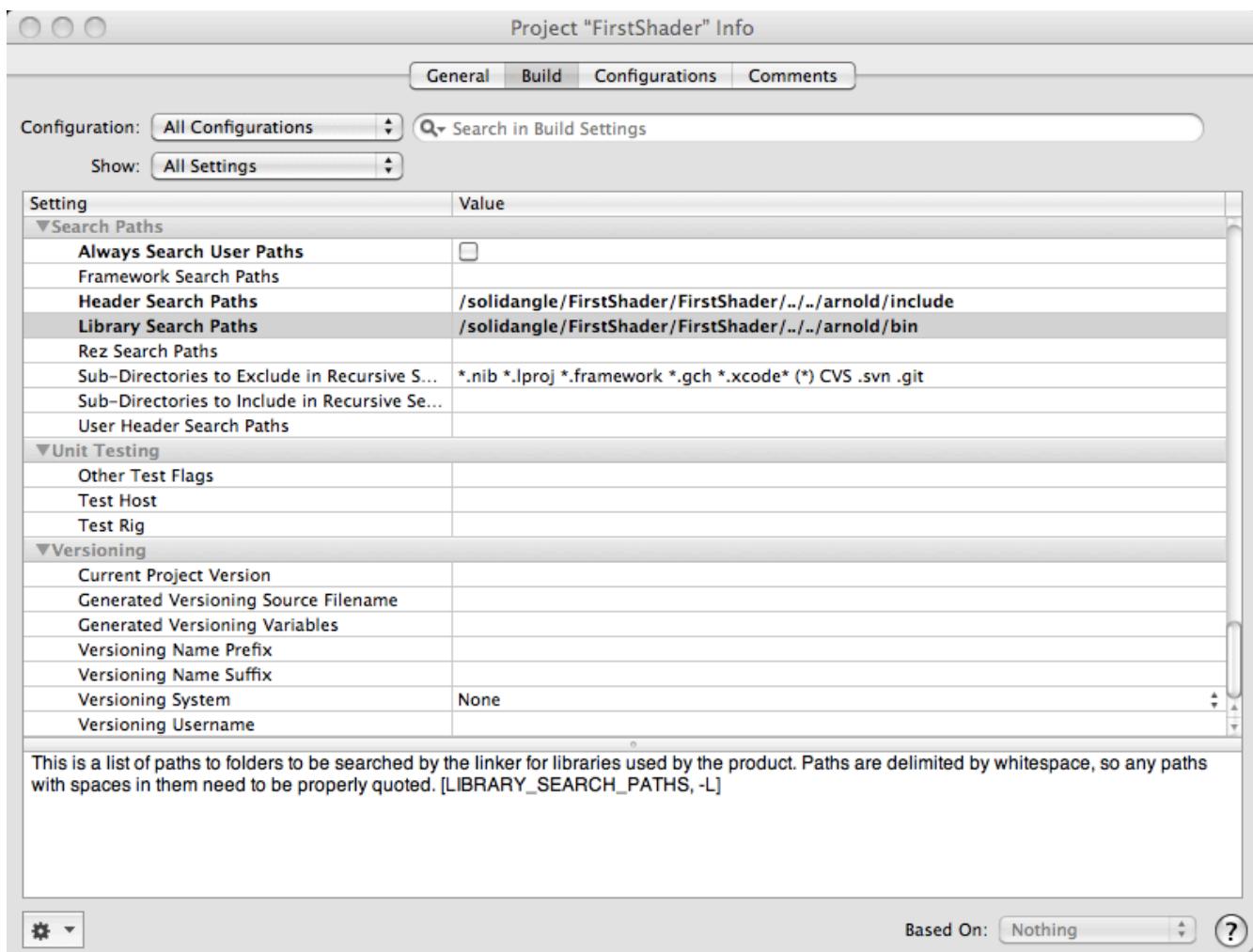
- Choose a Cocoa dynamic library:



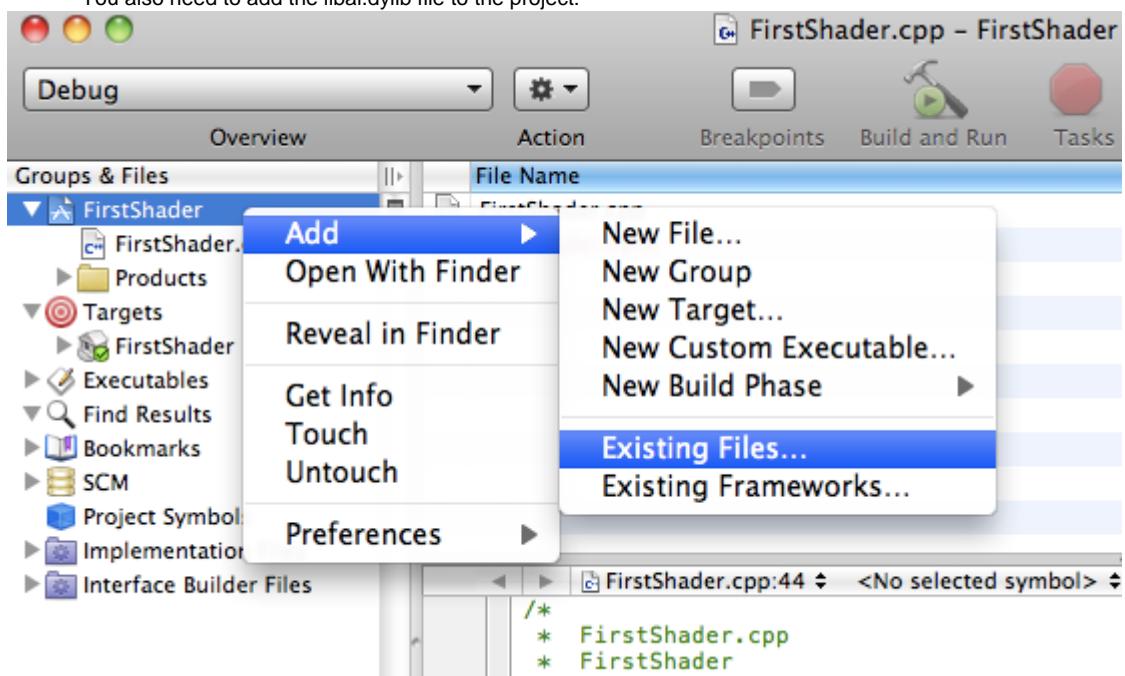
- You'll be asked to specify a name for the target:



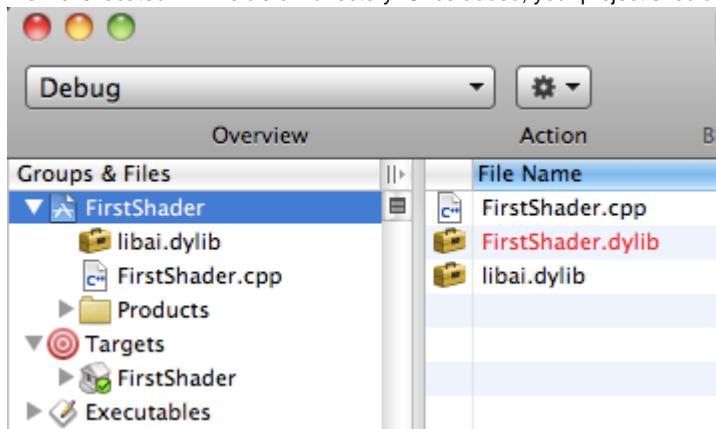
- You will then be prompted to choose project settings (you can edit project settings at a later stage from Xcode's project menu). You may need to scroll down to tell the project where to find the Arnold SDK include files and library files:



- You also need to add the libai.dylib file to the project:



The file is located in Arnold's bin directory. Once added, your project should look like this:



You should then be able to build the project and create a dynamic library. Once this dynamic library is placed where Arnold can find it you should be able to use it (for example place it in the bin directory under Arnold then type "kick -nodes" in the command-line).

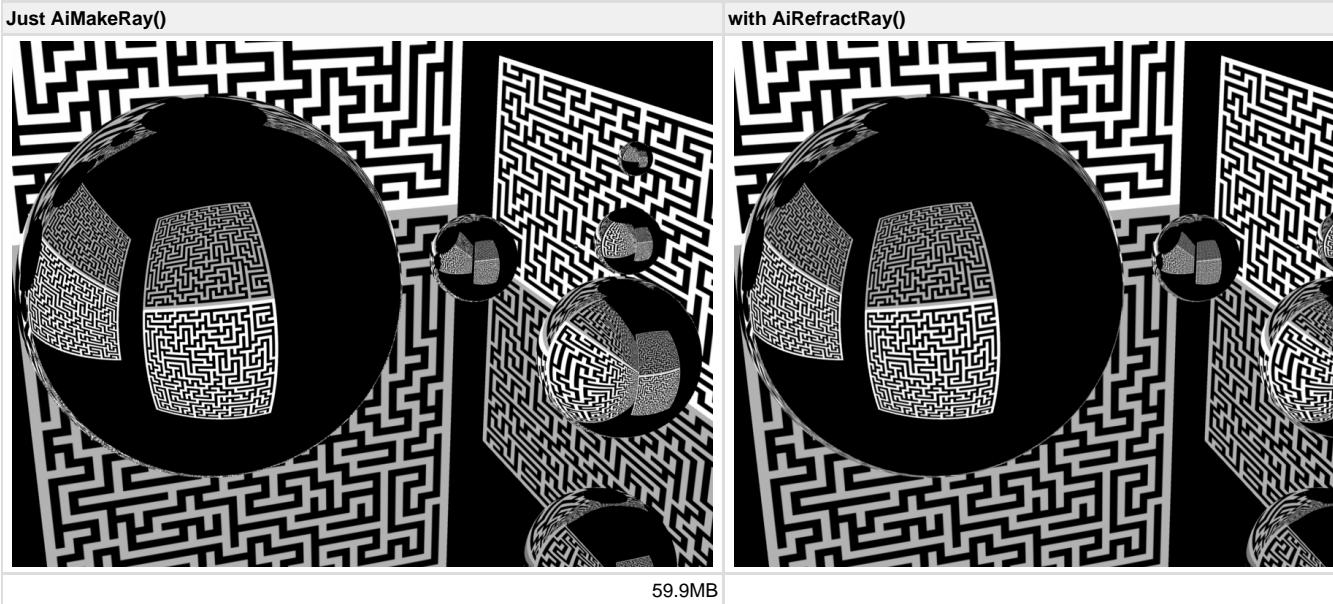
## Creating Refracted Rays With Proper Ray Derivatives

**AiMakeRay()** is able to properly set the ray direction derivatives (**ray->dDdx** and **ray->dDdy**) for reflection rays, but since it is not given the indices of refraction, it cannot correctly compute the derivatives for refraction rays. Instead, it just passes on the previous ray derivatives. This will result in worse texture quality and performance as the textures might end up overly blurred or aliased. In order to compute the proper ray derivatives, after **AiMakeRay()** is called, **AiRefractRay()** should be called which will compute the correct ray derivatives in addition to also computing the refracted ray direction.

Here's how we use this in the **standard** shader:

```
AtRay ray;
AiMakeRay(&ray, AI_RAY_REFRACTED, &sg->p, NULL, AI_BIG, sg);
const bool refracted = AiRefractRay(&ray, &sg->Nf, n1, n2, sg);
AiTrace(&ray, &sample);
```

Here is a simple example for what using **AiRefractRay()** provides. This was rendered with 2 AA\_samples and where each sphere is a refractive material. The amount of texture data read in is listed below the rendered image:



With **AiRefractRay()**, we have the correct ray derivatives and so are able to read from the correct texture mipmap level. This allows us to both remove the aliasing seen in the refracted textures through the spheres and cut down substantially on texture data read in. This should make it quite clear why it is important to use **AiRefractRay()** when casting refraction rays!

Note that in the case of reflection rays, **AiMakeRay()** is able to compute the correct reflection derivatives, so it is not necessary to call **AiReflectRay()**.

## Implementing MIS-sampled BRDFs

This is a short guide on how to modify an existing BRDF so that it takes advantage of the multiple importance sampling routines in Arnold. The conversion process fundamentally consists of two parts. The first part is the implementation of BRDF-specific functions, and the second is a series of changes that are required in the light sampling loop. To help illustrate this process, code snippets from the stretched-Phong BRDF of the **standard** shader will be used as an example.

### BRDF-specific functions

For the MIS sampling routines to work you need to implement 3 functions following the signatures defined by the **AtBRDFFevalSampleFunc**, **AtBRDFFevalBrdffunc** and **AtBRDFFevalPdfFunc** types in the file **ai\_shader\_brdff.h**. Below I will describe each of these in turn.

- **AtBRDFFevalSampleFunc** -- The implementation of this function generates a sampling direction given a pair of random variables (**rx** and **ry**) with a distribution whose probability density is the one described by the **AtBRDFFevalPpdfFunc** function. The implementation for the stretched-Phong BRDF used in the **standard** shader is as follows:

```
static AtVector PhongEvalSample(const void *brdf, float rx, float ry)
{
    const PhongBrdff *phong = (const PhongBrdff*)brdf;
    AtVector omega;

    omega.z = pow(rx, 1 / (phong->pexp + 1));
    float r = sqrtf(1 - SQR(omega.z));
    omega.x = r * cos(ry * AI_PITIMES2);
    omega.y = r * sin(ry * AI_PITIMES2);

    AiV3RotateToFrame(omega, phong->U, phong->V, phong->R);
    if (AiV3Dot(omega, phong->Ng) < 0)
        return AI_V3_ZERO;
    else
        return omega;
}
```

- **AtBRDFFevalBrdffunc** -- This function returns the result of your BRDF multiplied by the cosine of the angle between the surface normal and the incoming light direction (indir). The implementation in the **standard** shader is as follows:

```
static AtColor PhongEvalBrdff(const void *brdf, const AtVector *indir)
{
    const PhongBrdff *phong = (const PhongBrdff*)brdf;
    float reflectance = AiStretchedPhongBRDF(indir, &(phong->View), &(phong->Nf), phong->pexp);
    reflectance *= AiV3Dot(phong->Nf, *indir); // Revise MIS technique. BRDF shouldn't have to do this.
    return AiColor(reflectance);
}
```

- **AtBRDFFevalPpdfFunc** -- This function returns the probability density for a given direction. This function must match the density of samples generated by the **AtBRDFFevalSampleFunc** function, and it is required of every PDF that its integral over the sampling domain must be equal to 1. Its implementation for the stretched-Phong BRDF used in the **standard** shader is as follows:

```
static float PhongEvalPpdf(const void *brdf, const AtVector *indir)
{
    const PhongBrdff *phong = (const PhongBrdff*)brdf;

    if (AiV3Dot(phong->Ng, *indir) < 0)
        return 0;

    float RL = AiV3Dot(phong->R, *indir);
    RL = MAX(0, RL);
    return (phong->pexp + 1) * powf(RL, phong->pexp) * AI_ONEOVER2PI;
}
```

Note that all three of these functions receive as a parameter a void pointer. Think of it as the equivalent to the this pointer in a C++ class or to the local\_data of an Arnold shader. You can pass any extra data required of your functor implementations through this pointer.

## Light sampling loop modifications

The second modification that your shaders will require to take advantage of MIS is in the light sampling loop. Instead of directly evaluating your BRDF for each sample returned by **AiLightsGetSample()** you must instead use the **AiEvaluateLightSample()** function, passing it the shader globals, your newly defined functors and a pointer to any local data these functors may require. For example, the resulting stretched-Phong specular light loop in the standard shader looks like this:

```
PhongBrdf phong_brdf;
PhongBrdfInit(&phong_brdf, sg, clamped_pexp);

AiLightsPrepare(sg);
while (AiLightsGetSample(sg))
{
    if (AiLightGetAffectSpecular(sg->Lp))
        spec_dir += AiEvaluateLightSample(sg, &phong_brdf, PhongEvalSample, PhongEvalBrdf, PhongEvalPdf);
}
```

The PhongBrdfInit() function is where the local data required by the functors, PhongBrdf, is initialized. As you can see the lights need to be prepared and samples have to be requested just as in a conventional light loop, but instead of making calls to the BRDF and directly weighting your samples, they get weighted through the AiEvaluateLightSample() function call using the newly implemented methods.

Make sure that your new sampling loop does not discard light samples that are below the surface normal. This can result in noise or even darkening of the lighting estimate. Even though the light sample is below the surface, part of the light may actually be above the surface in the case of area lights so AiEvaluateLightSample() should be called anyway.

## Final notes

A few things to note is that our current MIS implementation takes an equal number of samples from the light towards the surface as from the surface towards the light and mixes them using the power heuristic. The total number of samples is controlled using the light's "samples" parameter. Also, please note that we are not entirely happy with the current MIS implementation and it may go through some refactoring at some point to simplify its adoption, improve its control and possibly to allow for other importance sampling techniques (resampling importance sampling, one-sample importance sampling, etc).

## Shaders Library

You can find in the arnoldpedia repository an [example set](#) of shaders. These shaders are all meant to be built into the same DSO, and there is one loader for the whole set (see shaderlib.cpp for the loader). These shaders complement the built-in set (such as `standard_surface`, `noise` and `ray_switch`) and thus focus on shader network connectors and helpful utilities.

These shaders are examples only or can be used as a basis upon which you may build your own shaders suitable for a shader network in UI applications such as The Foundry's Katana or in MtoA or SltmA. These do not ship with the Arnold core, and Solid Angle does not officially support them. You may find them useful, however.

- Color management: `blackbody`, `color_convert`, `color_correct`, `complement`, `mix`
- Math nodes: `abs`, `add`, `cross`, `divide`, `dot`, `exp`, `length`, `multiply`, `normalize`, `pow`, `reciprocal`, `subtract`
- Range management: `clamp`, `max`, `min`, `range`
- Type conversion: `rgb_to_float`, `rgb_to_vector`, `vector_to_rgb`
- User-data reading: `user_data_float`, `user_data_int`, `user_data_rgb`, `user_data_rgba`, `user_data_string`
- Utilities/misc: `cache`, `random`, `volume_collector`, `color_utils.h`, `memory.h`, `shaderlib.mtd`
- Katana .args files are found in the `Args` subdirectory

## Understanding GI With a Basic Shader

### Plastic Shader with GI

I assume you have read the [Writing Your First Shader](#) page, and understand the basics of compilation and installation of shaders. Here we present an old style "plastic" shader, and then show how this type of shader is properly written to use the features of global illumination.

First a refresher: the plastic shader in traditional scanline renderers uses a combination of 3 terms:

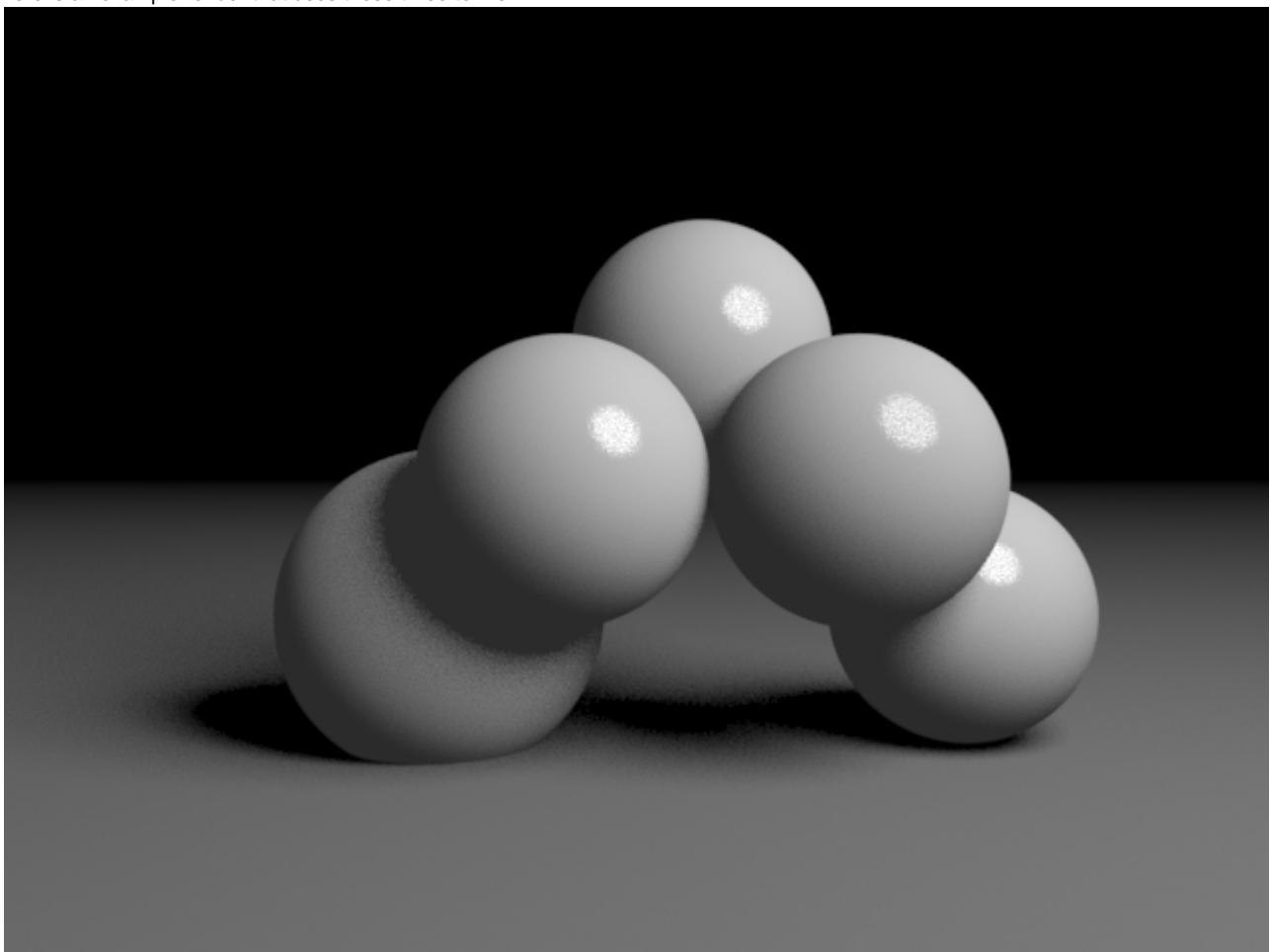
- Specular: the shiny reflected light that bounces off of shiny and glossy surfaces.
- Diffuse: the scattered surface color as lit directly by lightsources.
- Ambient: the ambient light bouncing around the scene, not directly from a lightsource.

You have probably been using these terms for some time, they are probably very familiar. Ambient is simply the user provided color, added evenly over the surface. Lambertian diffuse is the user supplied color multiplied by the light color, and then multiplied by the amount of light falling on the surface at whatever its angle is to the light (the cosine of the angle from the light to the normal, or  $L \cdot N$ ): [wikipedia](#). And a Blinn-Phong specular is the dot product between the normal and the halfangle from viewer to lightsource, raised to a power: [wikipedia](#).

The shader loops over every lightsource, accumulating the specular and diffuse components for each light, and then adds the ambient in at the end.

You can see bad speckly noise in the specular highlight from poor sampling; properly written BRDFs can take advantage of Arnold's multiple importance sampling (MIS) to fix this problem, see [Implementing Multiple Importance Sampled BRDFs](#).

Here is an example render that uses these three terms:



And here is the shader code for Arnold that created this image:

```

#include <ai.h>
#include <cstring>

AI_SHADER_NODE_EXPORT_METHODS(SimpleMethods);

enum SimpleParams {
    p_Ka_color,
    p_Kd_color,
    p_Ks_color,
    p_roughness
};

node_parameters
{
    AiParameterRGB("Ka_color", 0.7f, 0.7f, 0.7f);
    AiParameterRGB("Kd_color", 0.7f, 0.7f, 0.7f);
    AiParameterRGB("Ks_color", 0.7f, 0.7f, 0.7f);
    AiParameterFLT("roughness", 0.2f);
}

node_initialize
{
}

node_update
{
}

node_finish
{
}

shader_evaluate
{
    // classic plastic controls are: ambient, diffuse, specular, and roughness.
    // Ka (ambient color), Kd (diffuse color), Ks (specular color), and roughness (scalar)
    AtColor Ka = AiShaderEvalParamRGB(p_Ka_color);
    AtColor Kd = AiShaderEvalParamRGB(p_Kd_color);
    AtColor Ks = AiShaderEvalParamRGB(p_Ks_color);
    float roughness = 10 / AiShaderEvalParamFlt(p_roughness);

    AiLightsPrepare(sg);
    AtColor La = AI_RGB_BLACK; // initialize light accumulator to = 0
    while (AiLightsGetSample(sg)) // loop over the lights
    {
        float LdotN = AiV3Dot(sg->Ld, sg->Nf);
        if (LdotN < 0) LdotN = 0;
        AtVector H = AiV3Normalize(-sg->Rd + sg->Ld);
        float spec = AiV3Dot(sg->Nf, H); // N dot H
        if (spec < 0) spec = 0;
        // Lambertian diffuse
        La += sg->Li * sg->we * LdotN * Kd;
        // Blinn-Phong specular
        La += sg->Li * sg->we * pow(spec, roughness) * Ks;
    }
    // color = accumulated light + ambient
    sg->out.RGB = La + Ka;
}

node_loader
{
    if (i > 0) return false;

    node->methods      = SimpleMethods;
    node->output_type  = AI_TYPE_RGB;
    node->name         = "olde_plastic";
    node->node_type    = AI_NODE_SHADER;
    strcpy(node->version, AI_VERSION);
    return true;
}

```

With global illumination we can improve the image quality, which needs a few changes in how this shader is written.

First of all, we get rid of the ambient term. Most lighting pipelines have avoided using an ambient term for many years now, it often only flattens the image. Global illumination correctly computes the bounced light in the scene, making this old hack unnecessary.

The next concept to grasp is that the energy balance becomes important. Scanline renderers are a bit like 2d-compositing. The three terms are simply added up and mixed together. A global illumination framework is more like photography, bright surfaces reflect light energy more than dark ones, and affect the other objects in the scene. This means that the total energy your shader returns must be less than or equal to the energy in, unless it is meant to be a lightsource. In a global illumination system, light bounces off of multiple surfaces, and the shader is called each time. If your shader returns more energy than goes in, it can amplify the incoming light at every bounce. This feedback loop means even a tiny amount of extra energy can cause a scene to blow out past white very quickly. In this GI plastic shader, we account for this by adding a "specbalance" control. To get a brighter specular, you must take the energy from the diffuse, and vice-versa.

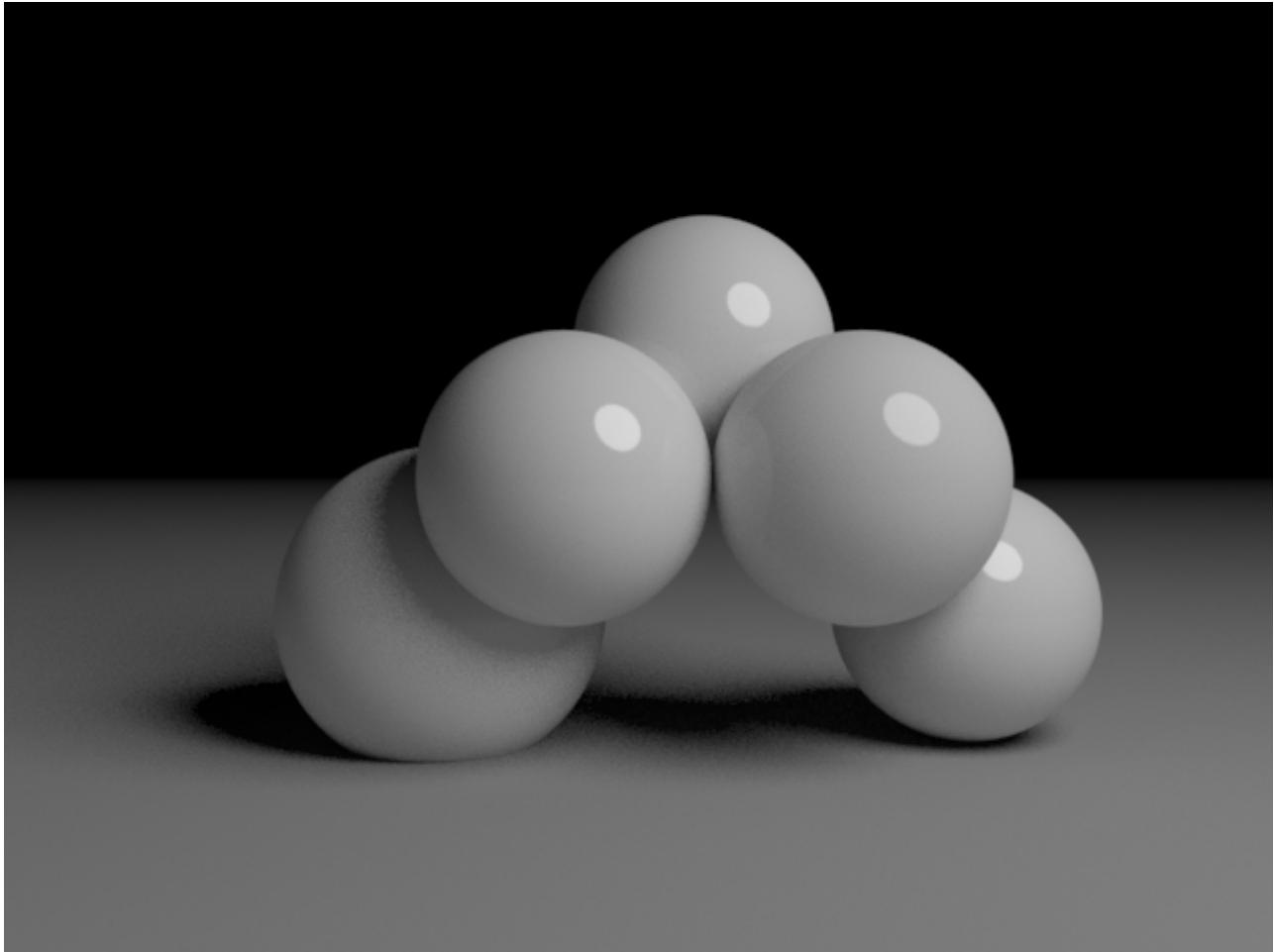
Getting good, energy balanced specular and diffuse terms can be a bit tricky; fortunately there are many good BRDFs that ship with Arnold's API, ready to be used. This shader takes advantage of Arnold's multiple importance sampling (MIS) methods, and splits the returned values into 4 terms:

- direct diffuse: old-style diffuse from direct light
- indirect diffuse: GI bounced light as it affects the diffuse surface computation
- direct specular: good energy-balanced specular from direct light
- indirect specular: GI bounced light as it affects the specular surface computation

To allow Arnold to take advantage of the multiple importance sampling technique, the BRDF must first be registered with an **MISCreateData**; the details of this is laid out in the [Implementing multiple importance sampled BRDFs](#) documentation.

In the lightloop, the direct light and specular are accumulated, and outside the loop the indirect functions are evaluated. This allows you to pass out the direct and indirect values to an AOV for compositing tweaks, while keeping the proper energy balance in the main render.

Here is an example render that uses global illumination (GI) and multiple importance sampling (MIS):



Note that, thanks to MIS, the specular highlight of the big area light renders with much less noise than before. And here is the shader code that created this image:



```

#include <ai.h>
#include <cstring>

AI_SHADER_NODE_EXPORT_METHODS(SimpleMethods);

enum SimpleParams
{
    p_Kd_color,
    p_Ks_color,
    p_roughness,
    p_specbalance
};

node_parameters
{
    AiParameterRGB("Kd_color", 1.f, 0.7f, 0.7f);
    AiParameterRGB("Ks_color", 0.7f, 0.7f, 0.7f);
    AiParameterFLT("roughness", 0.2f);
    AiParameterFLT("specbalance", 0.1f);
}

node_initialize
{
}

node_update
{
}

node_finish
{
}

shader_evaluate
{
    // Kd (diffuse color), Ks (specular color), and roughness (scalar)
    AtColor Kd = AiShaderEvalParamRGB(p_Kd_color);
    AtColor Ks = AiShaderEvalParamRGB(p_Ks_color);
    float roughness = AiShaderEvalParamFlt(p_roughness);
    float specbalance = AiShaderEvalParamFlt(p_specbalance);

    // direct specular and diffuse accumulators,
    // and indirect diffuse and specular accumulators...
    AtColor Dsa,Dda,IDs,IDD;
    Dsa = Dda = IDs = IDD = AI_RGB_BLACK;
    void *spec_data = AiWardDuerMISCreateData(sg, NULL, NULL, roughness, roughness);
    void *diff_data = AiOrenNayarMISCreateData(sg, 0.0f);
    AiLightsPrepare(sg);
    while (AiLightsGetSample(sg)) // loop over the lights to compute direct effects
    {
        // direct specular
        if (AiLightGetAffectSpecular(sg->Lp))
            Dsa += AiEvaluateLightSample(sg, spec_data, AiWardDuerMISSample, AiWardDuerMISBRDF, AiWardDuerMISPDF) *
specbalance;
        // direct diffuse
        if (AiLightGetAffectDiffuse(sg->Lp))
            Dda += AiEvaluateLightSample(sg, diff_data, AiOrenNayarMISSample, AiOrenNayarMISBRDF, AiOrenNayarMISPDF)
* (1-specbalance);
    }
    // indirect specular
    IDs = AiWardDuerIntegrate(&sg->Nf, sg, &sg->dPdu, &sg->dPdv, roughness, roughness) * specbalance;
    // indirect diffuse
    IDD = AiOrenNayarIntegrate(&sg->Nf, sg, 0.0f) * (1-specbalance);

    // add up indirect and direct contributions
    sg->out.RGB = Kd * (Dda + IDD) + Ks * (Dsa + IDs);
}

node_loader
{
    if (i > 0) return false;

    node->methods      = SimpleMethods;
    node->output_type = AI_TYPE_RGB;
    node->name         = "GI_plastic";
    node->node_type   = AI_NODE_SHADER;
    strcpy(node->version, AI_VERSION);
    return true;
}

```

}

The code, JPEG file, and a Scons SConstruct file is in the attached gzipped tarball.

## Using Trace Sets

We want to implement trace sets in an example shader. The shader has two string attributes: `reflection_set` and `shadow_set`. These define the name of the trace sets against which reflection and shadow rays will be traced.

There are two types of trace sets:

- When a trace set is *exclusive*, rays are traced against all geometry except the tagged nodes.
- When a trace set is *inclusive*, rays are traced against tagged nodes, but also against nodes that are not tagged at all (see example .ass below).

By convention, our shader will consider all trace sets as *inclusive*, except if the first character of the string is '`-`', in which case the rest of the name will be treated as an *exclusive* trace set.

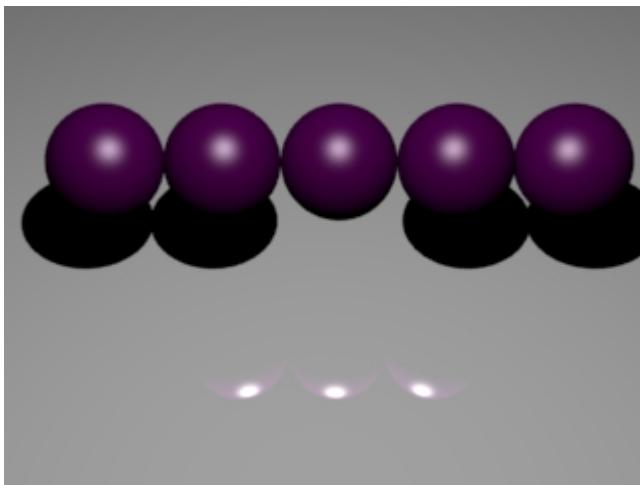
This shader could be implemented like this:

```
shader_evaluate
{
    ...

    // direct lighting (both diffuse & specular)
    AiShaderGlobalsSetTraceSet(sg, shadow_set, shadow_set_inclusive);
    AiLightsPrepare(sg);
    while (AiLightsGetSample(sg))
    {
        ...
    }
    AiShaderGlobalsUnsetTraceSet(sg);

    // mirror Reflection
    if (sg->Rr_refl < 2)
    {
        ...
        AiShaderGlobalsSetTraceSet(sg, reflection_set, reflection_set_inclusive);
        AiMakeRay(&ray, AI_RAY_REFLECTED, &sg->P, &reflected, AI_BIG, sg);
        AiShaderGlobalsUnsetTraceSet(sg);
        ...
    }
}
```

An example render and its .ass file that demonstrates the different trace set cases follows:



```
options
{
    AA_samples 3
    xres 320
```

```

yres 240
GI_diffuse_depth 0
}

persp_camera
{
    name mycamera
    position 0.5 5 -5
    look_at 0.5 0.2 0
    fov 38
    handedness left
}

point_light
{
    name mylight
    position 0.5 5 -2
    decay_type constant
}

standard
{
    name s1
    Kd 0.5
    Kd_color 1 0 1
    Ks 0.26
    specular_roughness 0.45
}

standard_tsets
{
    name s2
    Kd 0.5
    Kd_color 1 1 1
    Kr 1
    reflection_trace_set "reflection" # inclusive set
    shadow_trace_set "-shadow"      # exclusive set (this is parsed in the shader code)
}

plane
{
    name myplane
    shader s2
    normal 0 1 0
}

# "reflection" is used in the shader as an inclusive trace set,
#     so only geometry tagged with it will be traced
#
# "shadow" is used in the shader as an exclusive trace set,
#     so only geometry NOT tagged with it will be traced

# invisible in reflections, visible in shadows
sphere
{
    shader s1
    trace_sets 2 1 STRING "xxx" "yyy"
    radius 0.4
    center -1 1 0
}

# this is visible to both refl and shadows
sphere
{
    name centerSphere
    shader s1
    trace_sets 2 1 STRING "xxx" "reflection"
    radius 0.4
    center -0.2 1 0
}

# invisible to shadows, visible for reflection
sphere
{
    shader s1
    trace_sets 2 1 STRING "shadow" "reflection"
    radius 0.4
    center 0.6 1 0
}

```

```
}
```

```
# visible for shadows and reflections
# when no trace_set is assigned to a node, it is
# always visible to all rays
sphere
{
    shader s1
    radius 0.4
    center 1.4 1 0
}

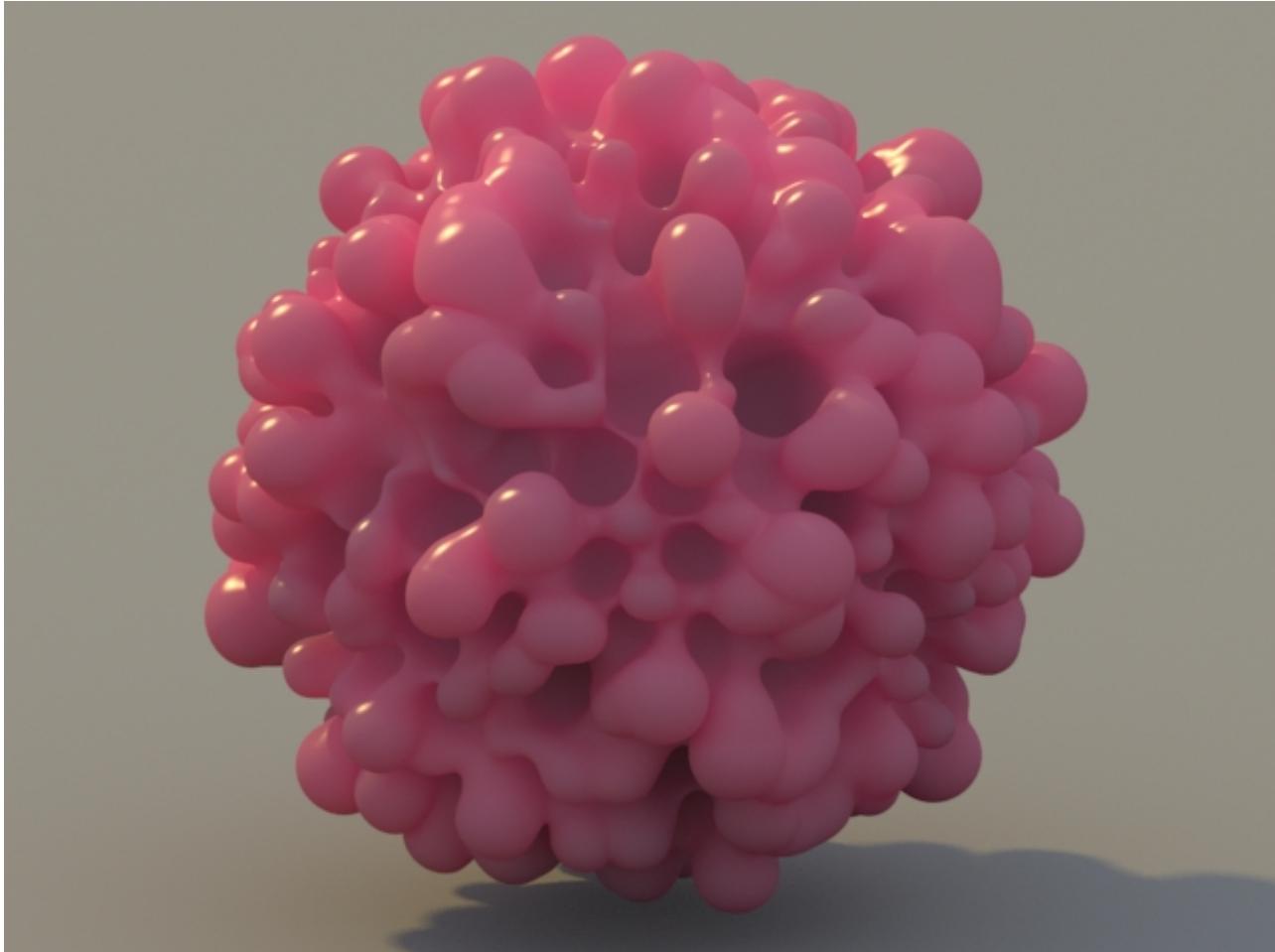
# visible for shadows, invisible for reflections
# if any trace set is specified (even a null one)
# the geometry will participate in the trace set
# mechanism (visible by default for exclusive sets,
# invisible by default for inclusive sets)
sphere
{
    shader s1
    trace_sets 1 1 STRING ""
```

```
radius 0.4
center 2.2 1 0
}
```

## Writing A Vector Displacement Shader

Arnold allows shaders to displace vertices in a polymesh node. The vertices are displaced in the direction and magnitude of the vector returned by the shader.

In this example shader, we compute a nonlinear displacement based on a noise function. This can displace along the normal of a surface like a traditional noise displacement shader, but it also has the option of blooming out the peaks and valleys of the displacement.



This is done by computing a noise function and its delta in the direction of two surface derivatives. This delta determines the amount the derivatives would be deflected by the displacement. By repeatedly deflecting the derivatives, we create a non-linear displacement that arcs out from the surface. This is equivalent to displacing the surface, computing the normal, and displacing again repeatedly.

Here is an animation showing the effect: [vimeo](#) / [download](#).

The shader source code:

```
/*
 * nonlinear noise displacement shader
 */

#include <ai.h>
#include <string.h>

#define _octaves      (params[0].INT)
#define _freq         (params[1].FLT)
#define _amplitude   (params[2].FLT)
```

```

#define _bloom      (params[3].FLT)
#define _type       (params[4].INT)

AI_SHADER_NODE_EXPORT_METHODS(NonlinNzMethods);

#define ENUM_SCALAR_TYPES { "perlin", "abs_perlin", "recursive", "abs_recursive",
NULL };
#define PERLIN      0
#define ABS_PERLIN   1
#define RECURSIVE    2
#define ABS_RECURSIVE 3
const char *types_enum[] = ENUM_SCALAR_TYPES;

node_parameters
{
    AiParameterINT ("octaves" , 3);
    AiParameterFLT ("freq"     , 1);
    AiParameterFLT ("amplitude", 1);
    AiParameterFLT ("bloom"    , 1);
    AiParameterENUM("type"     , PERLIN, types_enum);
}

float scalarfunc(AtPoint P, int type, int octaves)
{
    float doubler = 1;
    float NzAccum = 0;
    switch (type)
    {
        case PERLIN:
            return AiPerlin3(P);
        case ABS_PERLIN:
            return fabs(AiPerlin3(P));
        case RECURSIVE:
            for (int i = 0; i < octaves; i++) {
                NzAccum += AiPerlin3(P*doubler) / doubler;
                doubler *= 2;
            }
            return NzAccum;
        case ABS_RECURSIVE:
            for (int i = 0; i < octaves; i++) {
                NzAccum += fabs(AiPerlin3(P*doubler)) / doubler;
                doubler *= 2;
            }
            return NzAccum;
    }
    return AiPerlin3(P);
}

shader_evaluate
{
    const AtParamValue *params = AiNodeGetParams(node);

    AtPoint Ploc, Uloc, Vloc; // noise sample location, and over in U and V
locations
    float Np, Nu, Nv;          // noise at P, noise at location over in U and V
    float Udelt, Vdelt;        // delta in the noise over in U and V
    float delta = .01;         // distance delta for noise samples
}

```

```

AtVector U, V;
if (!AiV3IsZero(sg->dPdu) && !AiV3IsZero(sg->dPdv))
{
    // tangents available, use them
    U = sg->dPdu;
    V = sg->dPdv;
}
else
{
    // no tangents given, compute a pair
    AiBuildLocalFramePolar(&U, &V, &sg->N);
}
if (_type > ABS_PERLIN)
{
    // adjust delta to highest frequency in recursive noise
    delta *= pow(.5, _octaves) * 2;
}
Ploc = sg->Po * _freq;
Uloc = Ploc + U * delta;
Vloc = Ploc + V * delta;

// noise sampled at P, and over in U and V
Np = scalarfunc(Ploc, _type, _octaves);
Nu = scalarfunc(Uloc, _type, _octaves);
Nv = scalarfunc(Vloc, _type, _octaves);
Udelt = (Nu - Np) * _bloom;
Vdelt = (Nv - Np) * _bloom;
AtPoint Pstepped = sg->P;
int steps = 10;
float stepscale = _amplitude / steps;
for (int i = 0; i < steps; i++)
{
    // stepdir is the cross product of the derivatives
    AtVector stepdir = AiV3Cross(U,V);
    // deflect the derivatives
    U = AiV3Normalize(U + (stepdir * Udelt * stepscale));
    V = AiV3Normalize(V + (stepdir * Vdelt * stepscale));
    Pstepped += stepdir * Np * stepscale;
}
sg->out.VEC = Pstepped - sg->P;
}

node_initialize
{
}

node_update
{
}

node_finish
{
}

node_loader
{
}

```

```
if (i > 0) return FALSE;

node->methods      = NonlinNzMethods;
node->output_type   = AI_TYPE_VECTOR;
node->name          = "nonlinear_noise";
node->node_type     = AI_NODE_SHADER;
```

```

    strcpy(node->version, AI_VERSION);
    return TRUE;
}

```

And here is the .ass file that uses this shader:

```

options
{
    AA_samples 3
    xres 640
    yres 480
    background mysky
    GI_diffuse_depth 1
    GI_diffuse_samples 3
}

plane
{
    name myplane
    point 0 -8 0
    normal 0 1 0
    shader groundshader
}

polymesh
{
    name mysph
    nsides 6 1 UINT 4 4 4 4 4
    vidxs 24 1 UINT
    0 4 5 1 1 5 6 2 2 6 7 3 3 7 4 0 3 0 1 2 4 7 6 5
    vlist 8 1 b64POINT

AAB6wwAAAAAAHrDAAB6QwAAAAAAHrDAAB6QwAAAAAAHpDAAB6wwAAAAAAHpDAAB6wwAA+kMAAHrDAAB
6QwAA+kMAAHrDAAB6QwAA+kMAAHpDAAB6wwAA+kMAAHpD
    smoothing on
    subdiv_type catclark
    subdiv_iterations 7
    disp_map sphere_disp
    matrix
        0.94693 0 0.321439 0
        0 1 0 0
        -0.321439 0 0.94693 0
        0 0 0 1
    shader sphere_surf
    sss_sample_spacing 4
}

standard
{
    name sphere_surf
    Kd 0.15
    Kd_color 0.8 0.8 1
    Ks 0.1
    Ks_color 0.8 0.8 1
}
```

```
specular_roughness 0.3
Ksss 0.5
Ksss_color 1 0.05 0.2
sss_radius 80 80 80
}

nonlinear_noise
{
    name sphere_disp
    type perlin
    freq 0.025
    amplitude 80
    bloom 1
}

lambert
{
    name groundshader
    Kd_color 0.4 0.4 0.4
}

persp_camera
{
    name mycamera
    fov 11
    position 3677.0129 1039.1904 597.0592
    look_at 0 250 0
    up 0 1 0
}

point_light
{
    name key
    position -6000 10000 6000
    radius 400
    color 1 0.7 0.2
    intensity 2
}

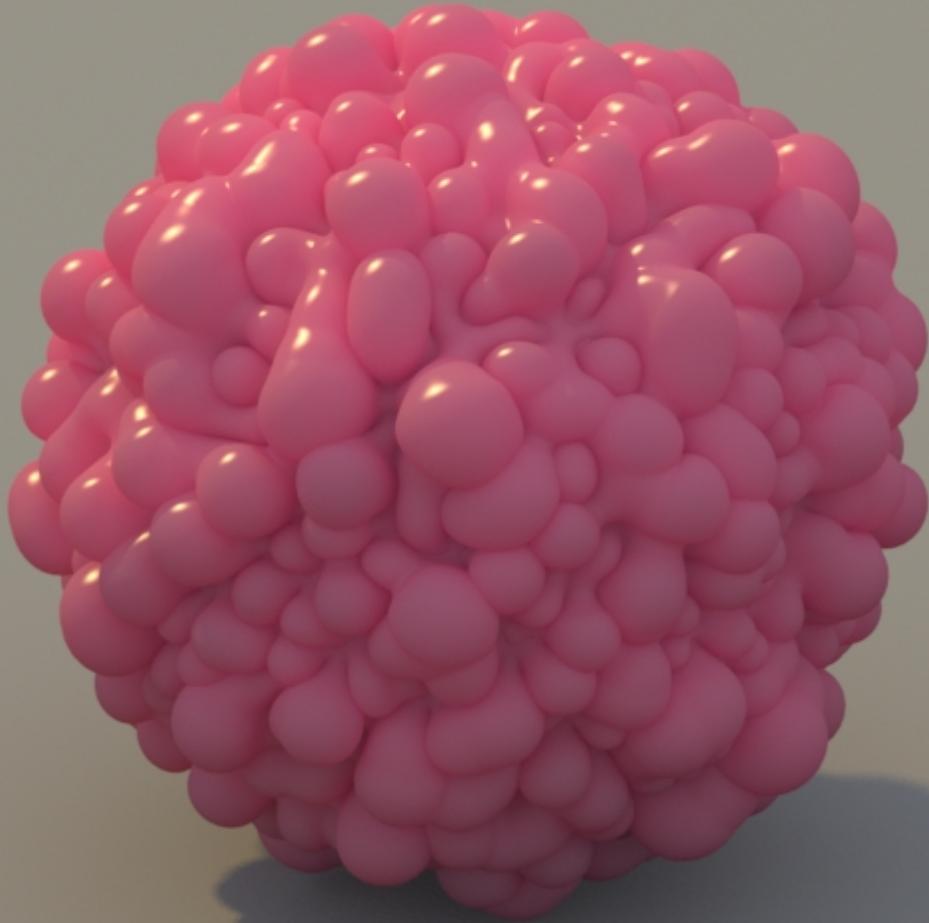
sky
{
    name mysky
```

```
    color 0.7 0.8 0.9  
    intensity 0.9  
}
```

Some examples of looks that can be obtained with this shader, along with the shader settings from the .ass file:

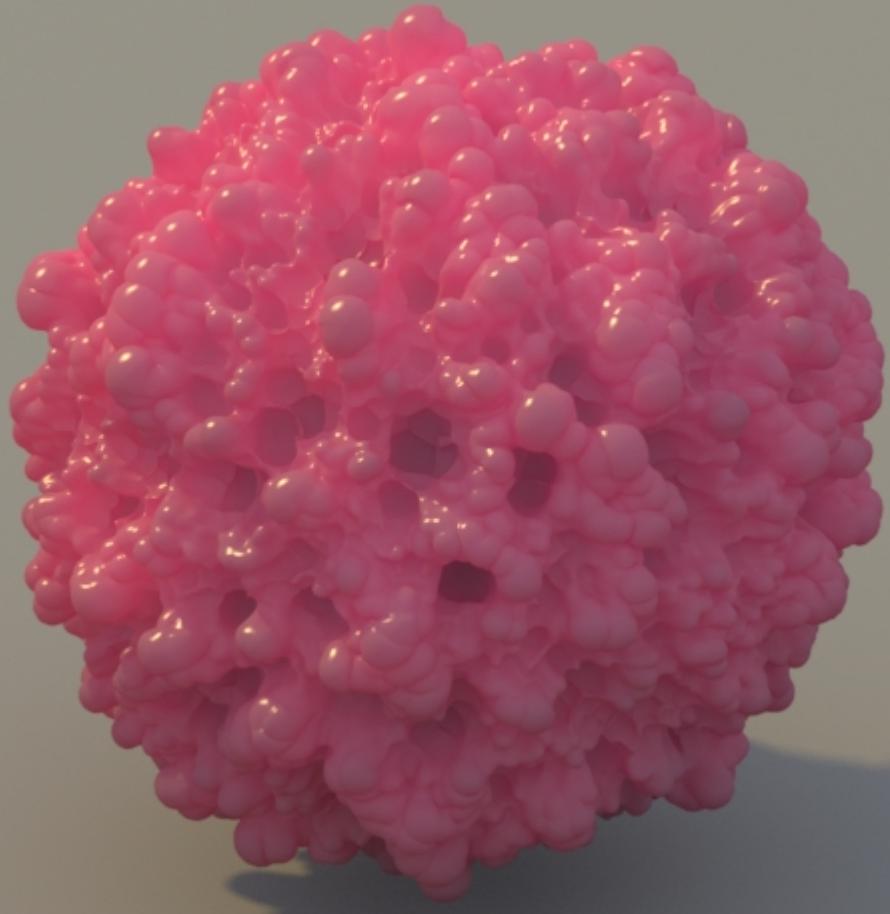
#### **Lumpy Nodules**

```
type abs_perlin  
freq 0.025  
amplitude 80
```



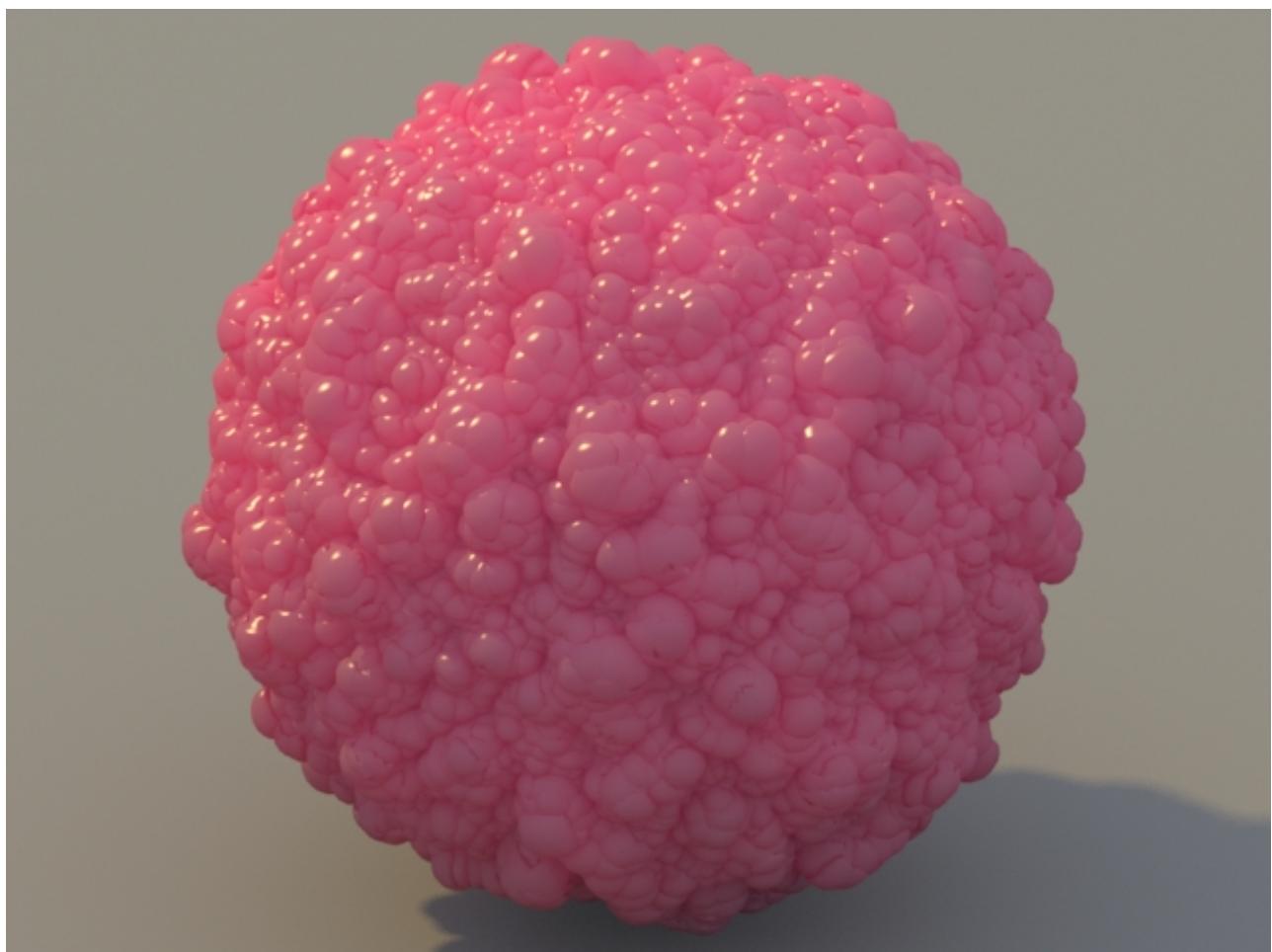
#### **Pomegranite**

```
type recursive
freq 0.025
amplitude 40
bloom 3
```



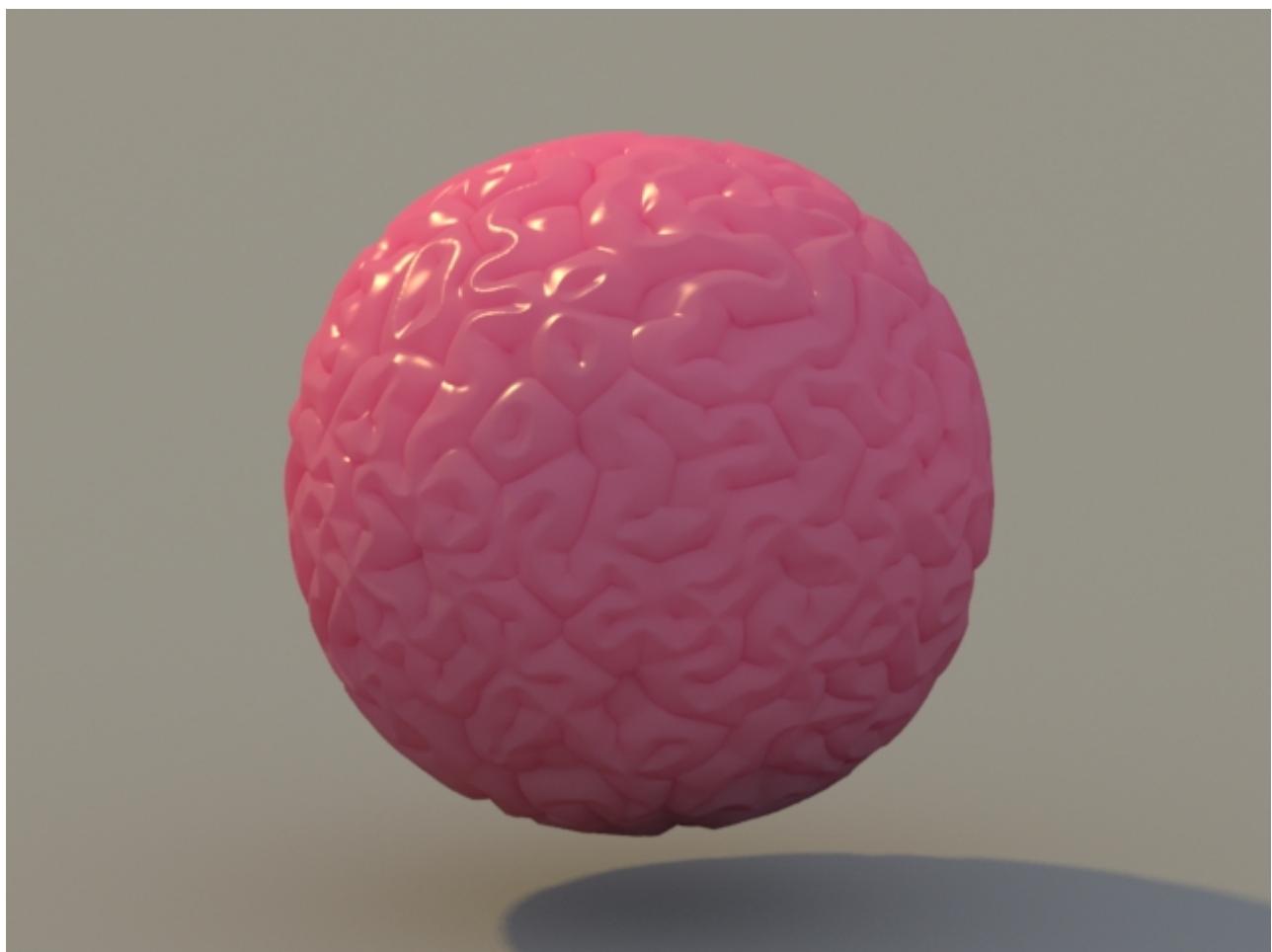
### **Cauliflower**

```
type abs_recursive
freq 0.025
amplitude 40
bloom 3
```



***Brain Coral***

```
type abs_perlin
freq 0.025
amplitude -40
bloom -4
```



## Writing Your First Shader

This is probably the most simple shader you can start playing with. It only outputs what you set in its **color** parameter.

```
#include <ai.h>

AI_SHADER_NODE_EXPORT_METHODS(SimpleMethods);

enum SimpleParams { p_color };

node_parameters
{
    AiParameterRGB("color", 0.7f, 0.7f, 0.7f);
}

node_initialize
{
}

node_update
{
}

node_finish
{
}

shader_evaluate
{
    sg->out.RGB = AiShaderEvalParamRGB(p_color);
}

node_loader
{
    if (i > 0)
        return false;

    node->methods      = SimpleMethods;
    node->output_type  = AI_TYPE_RGB;
    node->name         = "simple";
    node->node_type    = AI_NODE_SHADER;
    strcpy(node->version, AI_VERSION);
    return true;
}
```

### Short Description

**node\_initialize** is called only once for each instance of the shader, at the beginning of the first render.

**node\_update** is called once per render call (so, multiple times during progressive rendering or IPR) for each instance of the shader. And this includes the first render.

Summing up, for N calls to **AiRender()**, you have one call to **node\_initialize**, then N calls to **node\_update**. These numbers are not related to the number of threads at all.

**node\_finish** is called once for each instance of the shader when an **AiEnd()** is called.

### Compiling on Windows

Use the dummy project [here](#).

### Compiling on Linux

You'll need to compile each shader's source into an object file:

```
g++ -o shader1.os -c -fPIC -D_LINUX -I$ARNOLD_PATH/include
shader1.cpp
```

Keep in mind that you need a **node\_loader** entry point as explained [here](#).

Next, you need to generate a shared library:

```
g++ -o libmyshader.so -shared shader1.os -L$ARNOLD_PATH/bin  
-lai
```

Or if you want multiple shaders inside the library:

```
g++ -o libmyshader.so -shared shader1.os shader2.os shader3.os ... loader.os -L$ARNOLD_PATH/bin  
-lai
```

### ***Loading the shader***

Your shader library is now ready to use by any application that can load Arnold plugins, you'll just have to copy it somewhere it can be found by the application.

If you are using kick, you can specify the path with the -l option:

```
kick -l /path/to/plugins  
scene.ass
```

Note that there is no need to specify plugin filenames, **kick** will load all plugins that it can find in the plugins directory.

# API FAQ

## What is the difference between AiNodeGet\* and AiShaderEvalParam\*?

[Click here to expand...](#)

The main difference between evaluating a shader parameter via AiShaderEvalParam\* and evaluating via AiNodeGet\* is that the former supports shading networks, whereas AiNodeGet\* just returns the "static" parameter value without evaluating whatever other shader may be plugged in there. The other difference is that you can only callAiShaderEvalParam\* from within a shader's shader\_evaluate method, whereas you can call AiNodeGet\* anywhere.

Also, calls to AiShaderEvalParam\* have much lower overhead than equivalent calls to AiNodeGet\* because:

- AiNodeGet\* operates on strings which need slow string comparisons, hashing, etc.
- AiShaderEvalParam\* is optimized to work on shader nodes, whereas AiNodeGet\* can operate on any node types and thus requires more internal checks and book-keeping.

## How do I retrieve user-data from a shader?

[Click here to expand...](#)

In shader\_evaluate, use the AiUDataGet\* API, regardless of the storage qualifier of the user-data (constant, uniform or varying). Note that, for constant user-data, it is also possible (but very slow) to retrieve user-data with AiNodeGet\*. For performance reasons, you should never call AiNodeGet\* from a shader's shader\_evaluate method. It's OK to callAiNodeGet\* from node\_initialize and node\_update because these methods are only called once.

## What is the difference between AiNodeGet\* and AiUDataGet\*?

[Click here to expand...](#)

The AiNodeSet/Get\* API was designed for scene construction and manipulation, whereas the AiUDataGet\* API was designed to be called at shading time.

## Are shader parameter evaluations cached with AiShaderEvalParam\*?

[Click here to expand...](#)

Arnold does not cache any parameter evaluations (though this is scheduled for a future release). Evaluating the same parameter twice will evaluate the whole shading network twice, so care must be taken in shaders to avoid redundant evaluations.

## Can I use the AiNodeGetLink method inside a shader\_evaluate?

[Click here to expand...](#)

Technically, yes, you can call AiNodeGetLink(), AiNodeGetFlt() and similar APIs in shader\_evaluate. However, it is not recommended, because these calls are slow, involving several string comparisons, hash lookups and other potentially expensive operations that can severely affect render times. The solution is to precompute these calls in the node\_initialize or node\_update callbacks, which are executed once per session or per IPR pass, respectively, rather than per shading sample.

## Using Anisotropic Speculars

[Click here to expand...](#)

The microfacet BSDF has a separate roughness of the U and V tangent directions. If the roughness is the same, there will be anisotropic reflection. If they are different, there is an anisotropic reflection, and a tangent vector must be passed to the microfacet BSDF to indicate the direction of the U tangent. The V tangent direction is derived from the normal and V direction.

The simplest way to compute a tangent is using `sg->dpdu` which is the direction along the U axis of the UV coordinates. This lets users control the tangent direction by modifying the UV coordinates. If no UV coordinates are provided this vector will be zero, and we can fall back to an automatic tangent. For example, a tangent computed as if the shape is a sphere and the tangent follows the longitudinal direction.

```

AtVector U;
if ( !AiV3IsSmall(sg->dPdu) )
{
    // Tangent available from geometry and UV coordinates
    U = AiV3Normalize(sg->dPdu);
}
else
{
    // Automatic polar tangent fallback
    AtVector unused_V;
    AiV3BuildLocalFramePolar(U, unused_V, normal);
}

```

The above technique combined with a "rotation" map is usually enough to get anisotropy oriented any way you want, though it is possible to use custom tangents from user data.

### How do the visibility and the sidedness params of a node work?

▼ [Click here to expand...](#)

They are masks defined with the following values:

Ray type	Hex value	Description
AI_RAY_UNDEFINED	0x00	undefined type
AI_RAY_CAMERA	0x01	ray originating at the camera
AI_RAY_SHADOW	0x02	shadow ray towards a light source
AI_RAY_DIFFUSE_REFLECT	0x20	indirect diffuse reflection ray
AI_RAY_DIFFUSE_TRANSMIT	0x04	indirect diffuse transmission ray
AI_RAY_SPECULAR_REFLECT	0x50	indirect specular reflection ray
AI_RAY_SPECULAR_TRANSMIT	0x08	indirect specular transmission ray
AI_RAY_VOLUME	0x10	indirect volume scattering ray
AI_RAY_SUBSURFACE	0x80	subsurface scattering probe ray (for internal use)
AI_RAY_ALL	0xFF	mask for all ray types

By default, both visibility and sidedness are set to 255 (0xFF). So, if you want a polymesh to be visible only for camera rays, its visibility value would be AI\_RAY\_CAMERA. And if you want it to be visible to Camera and Reflection rays you would do AI\_RAY\_CAMERA + AI\_RAY\_DIFFUSE\_REFLECT + AI\_RAY\_SPECULAR\_TRANSMIT.

### What values does the disp\_map polymesh parameter support?

▼ [Click here to expand...](#)

If the shader connected to the parameter outputs a float value, then displacement happens along the normal to the surface. But if it outputs a 3D vector, that will be the displacement vector itself, in object-coordinates

There is no native support for tangent-space displacement vectors in Arnold. However, you should be able to implement a displacement shader that interprets incoming tangent-space vectors and transforms them on-the-fly to the coordinate space defined by (N, dPdu, dPdv).

### How can I attach user data to a ray that can be queried in subsequent ray hits?

▼ [Click here to expand...](#)

You can attach arbitrary data to the shading state that's passed down the ray tree using the message passing mechanism. For example:

```

// To set a value:
float IOR = 1.2f;
AiStateSetMsgFlt("IOR", IOR);
...

// To get a value
float IOR;
if (AiStateGetMsgFlt("IOR", &IOR))
{
...
}

```

## How does the alpha output of a shader connected to options.background affect the beauty pass?

[Click here to expand...](#)

The alpha component of the beauty pass represents the cumulative opacity of a pixel, which is determined by the out\_opacity shader global and not the "A" component of an RGBA shader's result.

## How to fill the different curves parameters?

[Click here to expand...](#)

The number of varying values for a given curve is based on the following formulae:

- num\_segments = (num\_points - 4)/vstep + 1
- num\_varying = num\_segments + 1

Step size (vstep) for each of the curve bases:

- bezier: 3
- b-spline: 1
- catmull-rom: 1

Essentially, there is one varying value at the start of each segment, with one extra value at the end of the curve. This is so that the varying values are interpolated smoothly from the start to the end of each, being continuous with the next and previous segments. Each curve basis defines its number of segments differently based on the total number of control points, hence the differing stepsizes above.

A couple of examples:

- You have a bezier-basis curve you want to specify with 13 control points. This means there are  $(13-4)/3+1 = 4$  bezier segments, and so you need 5 radii.
- You have a B-spline or Catmull-Rom curve you want to specify with 14 control points (something you can't do with beziers, which always have the number of control points be a multiple of 3 + 1). This means there are  $(14-4)/1+1 = 11$  segments, so you need 12 radii.

## What happens if a polymesh node in a nass file has non sides array defined?

[Click here to expand...](#)

If nsides is not defined, Arnold considers the polymesh node to be a triangular mesh. You can also define a single value instead of an array and Arnold will understand every face is of that number of vertices.

## How to declare user data parameters in a node?

[Click here to expand...](#)

To add a user-defined parameter in an assfile, you need to first declare it like AiNodeDeclare() does in the API. This can be done for all node types and is pretty useful when creating a procedural DSO that reads global information from the options block or the current camera. It's similar to the Renderman RiAttribute and RiOptions calls; the only thing is since Arnold isn't a state machine you might have to duplicate data on your procedural.

The declare line takes similar parameters to AiNodeDeclare(node, name, declaration):

- node is implicit and maps to the currently declared node.
- name is the user-parameter name as first argument

- declaration, isaclassandatype.
  - class := { constant, uniform, varying }
  - type := { BYTE, INT, BOOL, FLOAT, RGB, POINT, STRING, etc. }

After declaration, the parameter can be set as if it was a normal parameter.

```
options
{
  ...
declare my_user_param constant STRING
my_user_param "I want to render Rainbows with Unicorns."
...
}
```

## Do search paths support environment variable expansion?

[Click here to expand...](#)

The texture\_searchpath, procedural\_searchpath and plugin\_searchpath options support the expansion of environment variables delimited by square brackets. This works both using the API and inside .ass files as shown below:

```
AtNode *options = AiUniverseGetOptions();
AiNodeSetStr(options, "procedural_searchpath", "[MY_ENVAR_PATH]/to/somewhere");
```

```
options
{
  AA_samples 4
  GI_diffuse_samples 4
  ...
  procedural_searchpath "[MY_ENVAR_PATH]/to/somewhere"
}
```

## Is there a way to view the output image of a render in progress?

[Click here to expand...](#)

There is a workaround to see the progress of a render outside a general display driver.

- Set your output to tiledexr, zip is fine.
- Set the bucket scanning method to 'top'.
- Use a viewer like imf\_disp.

## What parameters support motion blur keys?

[Click here to expand...](#)

Geometry:

- For all shape nodes: matrix.
- For the polymesh node: vlist and nlist.
- For the curves node: control points, radius, and orientations.
- For the points node: points, radius, aspect, rotation.

Cameras:

- matrix and associated positional/orientation attributes like position, look\_at, up.

Lights:

- matrix and associated positional attributes: position, look\_at, up, direction.

## Shaders:

- Nothing is interpolated in the built-in Arnold shaders, with the exception of gobo.rotate

**Is there any difference between having subdiv\_type = none and subdiv\_type = linear when subdiv\_iterations is set to zero in a polymesh?**

▼ [Click here to expand...](#)

On polymesh nodes with subdiv\_type set to linear and subdiv\_iterations set to zero, the polymesh will ignore its provided set of user-defined shading normals and will issue a warning if they exist. On polymesh nodes with subdiv\_type set to none and subdiv\_iterations set to zero, normals defined by the user will not be ignored. This means the output result could change quite a bit if the user is ever providing normals for the polymesh it generates.

**How is the Z-depth AOV stored?**

▼ [Click here to expand...](#)

The Z AOV will have non-normalized depth data in the alpha channel between the near and far camera clipping planes, using the plane distance by default.

**Does min\_pixel\_width apply to all rays?**

▼ [Click here to expand...](#)

`min_pixel_width` applies to the hair as a preprocessing step so that it will affect all intersections. When applying stochastic opacity, the hair opacity is automatically reduced to compensate for increased curve radii. See the [Closures](#) documentation for how to use stochastic transparency in shaders.

Since transparent hairs are more expensive to render, `min_pixel_width` will render each AA samples slower. However, the reduction in noise will usually reduce overall render time.

`min_pixel_width` is applied to only camera rays (not shadow rays) for hair and for camera rays and shadow rays for points.

**Parameter Connectivity rules**

▼ [Click here to expand...](#)

We support connecting any individual component of an output to an input or an input component. These are all valid connections in a .ass

```
base_color myshader      # link RGB to RGB
base_color.r myshader.b  # link B channel to only R channel
base_color myshader.b    # link B channel to RGB channels
base_color.r myshader    # average RGB values and link to only R channel
```

We also support array element linking, thus linking a shader output with a specific array element in the input array. We don't support multiple outputs.

# Arnold Scene Source (.ass)

## The Structure of .ass Files

Arnold's native scene description file format, known as ".ass" files, are human-readable ASCII text files. An .ass file typically contains cameras, lights, model geometry and shaders as a list of nodes with their connections and parameters.

The Arnold DCC plug-ins for Maya, Houdini, etc. can export ass files. Arnold's command line renderer, `kick`, can be used to render these ass files into image files.

## An example .ass file

The following is a simple .ass file that contains an options block, a filter node, a driver, a camera, a light, a polymesh and a shader.

```
options
{
    AA_samples 3
    outputs "RGBA RGBA myfilter mydriver"
    xres 720
    yres 486
}
gaussian_filter
{
    name myfilter
    width 2.0
}
driver_tiff
{
    name mydriver
    filename "image.tif"
    color_space auto
}

persp_camera
{
    name mycamera
    fov 53.638
    matrix
    1 0 -0 0
    -0 0.995 -0.0995 0
    0 0.0995 0.995 0
    0 2 20 1
}

distant_light
{
    name mylight
    matrix
    0.78867512 -0.21132487 -0.57735025 0
    -0.21132487 0.78867512 -0.57735025 0
    0.57735025 0.57735025 0.57735025 0
    1 1 1 1
    color 1 1 1
    intensity 1
    cast_shadows on
}
```

```
polymesh
{
    name mysphere
    nsides 6 1 BYTE 3 3 3 3 3 3
    vidxs 18 1 UINT
    3 2 0 2 3 1 4 3 0 3 4 1 2 4 0 4 2 1
    nidxs 18 1 UINT
    0 0 1 0 0 2 3 3 4 3 3 5 6 6 7 6 6 8
    vlist 5 1 VECTOR 0 -4 0 0 4 0 -4 0 0 2 0 3.4641015
    2 0 -3.4641015
    nlist 9 1 VECTOR
    -0.5 0 0.8660254 -0.44721359 -0.44721359 0.77459669
    -0.44721359 0.44721359 0.77459669 1 0 0
    0.89442718 -0.44721359 0 0.89442718 0.44721359 0
    -0.5 0 -0.8660254 -0.44721359 -0.44721359 -0.77459669
    -0.44721359 0.44721359 -0.77459669
    smoothing on
    matrix
    1 0 0 0
    0 1 0 0
    0 0 1 0
    0 0 0 1
    shader myshader
}

standard_surface
{
    name myshader
    base 0.7
    base_color 0 1 0
    specular 0.05
```

```
specular_color 1 1 1
specular_roughness 0.3
}
```

Just like in Python, one-line comments begin with the '#' character.

## Nodes and Parameters

Arnold is built around different types of nodes. For example, there are shader nodes, camera nodes, light nodes, filter nodes, geometry (aka "shape") nodes and output driver nodes. Each node has a number of parameters.

Each type of node is identified by a unique name (options, persp\_camera, polymesh, lambert, etc.). You can have as many nodes of each node type as you want. Each node is uniquely identified by a string parameter called "name". To create a node, you write the node type and enclose the definition of the parameter values within brackets. For example:

```
lambert
{
    name my_white_shader
    Kd 0.9
    Kd_color 1 1 1
}
```

## Nodes

Here are some of the more important nodes that can be found in an ass file.

- **options** - This node is a container of global rendering options, for example:
  - **xres, yres**: Image resolution
  - **AA\_samples**: Antialiasing samples
  - **camera**: the active camera. It must point to a valid camera node in the scene
  - **outputs**: This array of strings defines a mapping of AOV channels (or layers) to output drivers. The format of each of the strings is: "<AOV\_name> <data\_type><filter> <driver>"
- **gaussian\_filter** - This is the default type of filter used by **kick**. The filter has a user-specified width which defaults to 2.0 pixels.
- **driver\_tiff** - Driver node that can be referenced by the **outputs** parameter in the **options** node. It has a **filename** parameter for the output file that will store the final rendered image.
- **persp\_camera** - Camera node that can be referenced by the **camera** parameter in the **options** node. Among other parameters, it has a field-of-view and a camera-to-world matrix that defines the orientation.
- **distant\_light** - Distant (or directional) light node with a transformation matrix, color, intensity, etc.
- **polymesh** - This is the most important geometric primitive in the renderer. Some of its parameters are:
  - **vidxs, nidxs, vlist, nlist**: These arrays describe the mesh vertices, normals, and their respective topologies (face indices). If the mesh has UV coordinates, they would be stored in the **vidxs** and **uvlist** parameters.
  - **visibility, sidedness**: These are bitmasks that define the visibility and sidedness properties for each ray type (camera, shadow, reflection, etc.).
  - **matrix**: The object-to-world transformation matrix of the mesh.
  - **shader**: Pointer to the shader node that will be executed when shading the object

## Parameters

All parameters have a default value, so you don't have to explicitly set all of the available parameters. Default parameter values can be queried with Arnold's command line renderer, **kick**. For example, to find the default value of the parameter **Kd** in the **lambert** shader, type this:

```
% kick -info lambert.Kd
node: lambert
param: Kd
type: FLOAT
default: 0.7
```

The most common types are BOOL, INT, UINT, ENUM, FLOAT, MATRIX, VECTOR, NODE and STRING.

## Arrays

Arrays of a basic type, e.g. VECTOR[ ] or FLOAT[ ], are specified with the following syntax:

```
parameter_name <num_elements> <num_motionblur_keys> <data_type> <elem1> <elem2>
<elem3> <elem4> ...
```

For example, the polymesh node has a parameter called vlist, the array of points where the polygon vertices are stored:

```
% kick -info polymesh.vlist
node: polymesh
param: vlist
type: VECTOR[ ]
default: (empty)
```

A polymesh with a single triangle would, therefore, be specified with an array of three VECTORS:

```
polymesh
{
...
vlist 3 1 VECTOR 0 0 0 1 0 0 0 0 0 1
...
}
```

## Motion Blur

For parameters that support motion blur, you can define several values for each of the motion blur time samples (or "keys"). The following example shows the list of vertices for one triangle with two motion keys, where the triangle has moved 5 units in the Y direction:

```
vlist 3 2 VECTOR 0 0 0 1 0 0 0 0 1 0 5 0 1 5 0 0 5 1
```

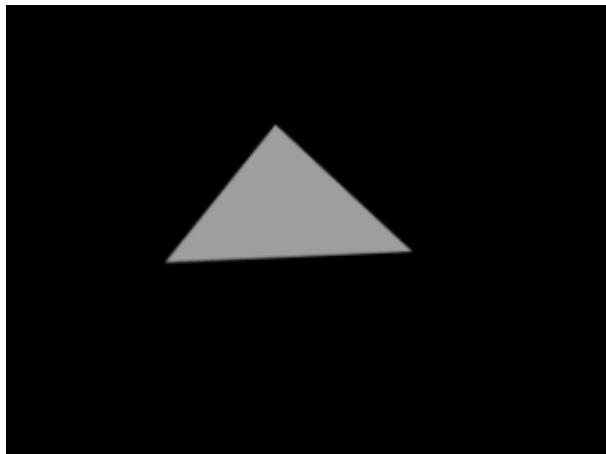
The same vertical motion can be achieved by storing static triangle vertices and providing multiple transformation matrices instead:

```
vlist 3 1 VECTOR 0 0 0 1 0 0 0 0 0 1
matrix 1 2 MATRIX
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
0 1 0 0
0 0 1 0
0 5 0 1
```

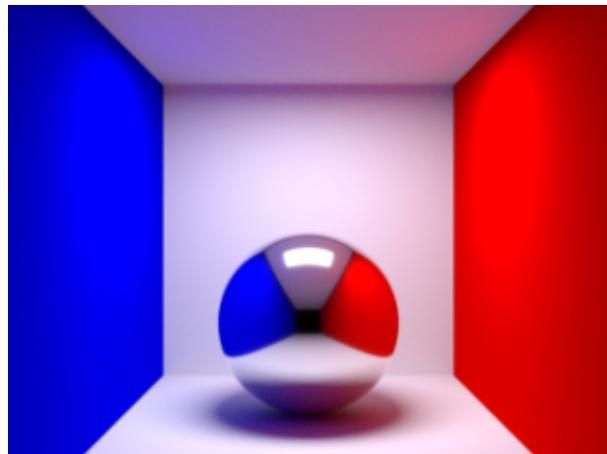
- [.ass File Examples](#)

## .ass File Examples

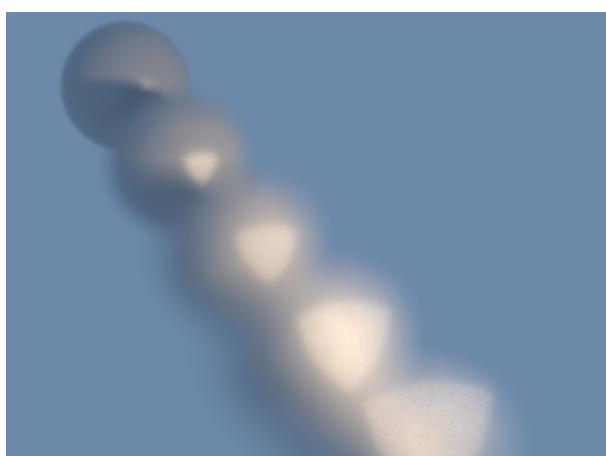
These files can be rendered using Arnold's command line renderer 'kick'. More information can be found [here](#).



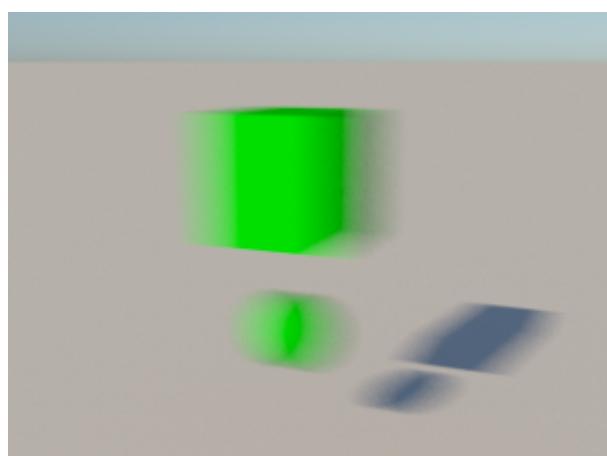
Very simple scene with a gray triangle with one light.



Classic Cornell box



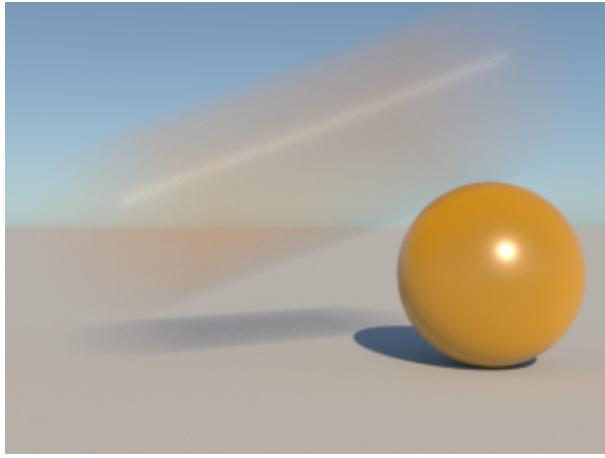
Depth of field with 3 aperture blades, producing a bokeh effect



Two polymeshes with deformation motion blur (2 motion keys)

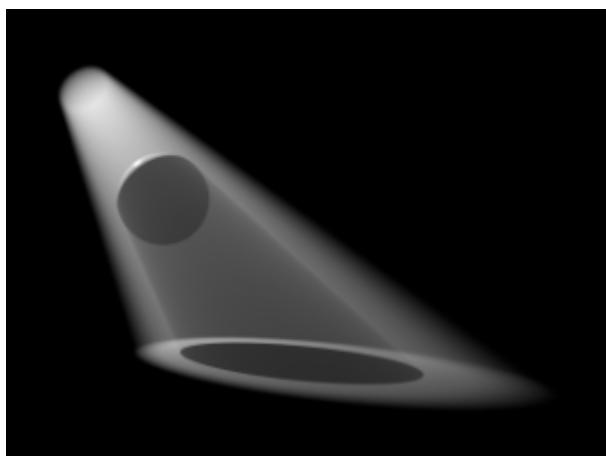
[Download .ass file](#)

[Download .ass file](#)



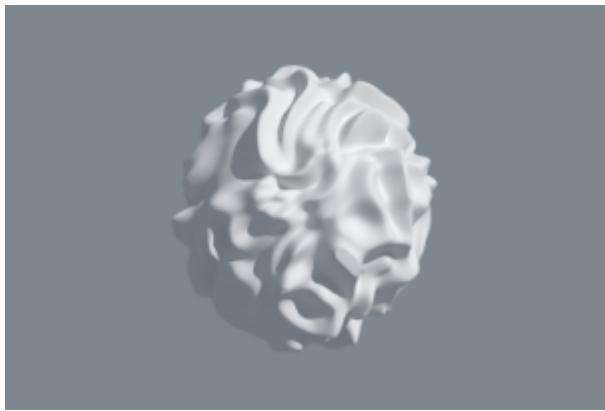
Sphere primitive with transformation motion blur

[Download .ass file](#)



Volumetric scattering from a spotlight with enlarged lens radius

[Download .ass file](#)



Displacement with a procedural noise

[Download .ass file](#)

# Command Line Rendering (kick)

## Introduction

Arnold's command line renderer is called kick. This reads an [.ass file](#), renders the scene using Arnold and outputs an image file. Kick can also be used to query Arnold nodes for their parameters and default values. It can also be used for scene debugging.

Kick is included in the plug-in downloads. After installation, you will find it in the bin subdirectory of the Arnold distribution.

## Running Kick

To run kick, first, open a shell (terminal). Here's an example on Windows OS.

kick always loads shaders and procedurals from the current directory, so don't run kick in a folder with lots of other DLLs/SOs/DYLIBs. kick will try to load each DLL/SO/DYLIB, to check whether it contains shaders or procedurals.

```
$ set ARNOLD_BIN_PATH=C:\solidangle\Arnold-5.0.0.0\bin  
$ %ARNOLD_BIN_PATH%\kick  
Arnold 5.0.0.0 [2cfbe09c] linux clang-3.9.1 olio-1.7.12 osl-1.8.2 vdb-4.0.0  
rlm-12.2.2 2017/04/10 16:48:44  
No arguments. Try kick --help for a command summary
```

## Useful Commands

To try out the following commands download one of the files like [cornell.ass](#) from the [example ass files page](#).

One of the most useful commands is "-h" or "--help". This will display a list of all the available options in kick:

```
kick -h
```

Use the "-i" option to read an .ass file and render it:

```
kick -i path/to/cornell.ass
```

Note that the "-i" option is not strictly necessary, kick automatically recognizes arguments ending in .ass, so this works too:

```
kick path/to/cornell.ass
```

By default, a window will pop up displaying the image as it's being rendered. You can turn off the display window with the '-dw' option:

```
kick cornell.ass -dw
```

If the geometry is colored pink, it could be because the path to the shaders may be required. This can be added using the "-l" flag:

```
kick cornell.ass -l /path/to/plugin/shaders/
```

Log information is sent to stdout. You can increase or decrease the log verbosity with the "-v <n>" option (default verbosity is 1). The most verbose option is "-v 6":

```
kick cornell.ass -v 2
```

Switch off the log output with "-v 0":

```
kick cornell.ass -v 0
```

To save the rendered image in an output file use the "-o" flag:

```
kick cornell.ass -o cornell.exr
```

Change the render display size with the "-r <width> <height>" option:

```
kick cornell.ass -r 1024 720
```

Print the Arnold version number, or the entire version string:

```
kick -av  
kick --version
```

Print diagnostic information about the license servers and list installed licenses available and in use:

```
kick -licensecheck
```

To override the antialiasing samples:

```
kick cornell.ass -as 3
```

To override the diffuse GI samples:

```
kick cornell.ass -ds 3
```

To disable the progressive refinement mode:

```
kick cornell.ass -dp
```

For debugging purposes you can globally disable several features like textures, lights, and shaders, motion blur, subdivs, displacement or SSS:

```
kick cornell.ass -it
kick cornell.ass -il
kick cornell.ass -is
kick cornell.ass -imb -isd -idisp -isss
```

You can install custom Arnold nodes by loading them from a dynamic library (.dll or .so) with:

```
kick cornell.ass -l path\to\plugin -l path\to\more\plugins
```

You can also get a list of all the installed nodes (both built-in and dynamically-loaded) with:

```
kick -nodes
kick -l path\to\plugins -nodes
```

You can inspect nodes with "-info":

```
kick -info polymesh
kick -info options
kick -l path\to\plugins -info custom_plugin_node
```

Or get more information about a given parameter:

```
kick -info polymesh.sidedness  
kick -info options.bucket_scanning
```

Override any parameter of any node using the "-set" command:

```
kick cornell.ass -set options.AA_samples 3
```

Override any parameter of all nodes of a particular type:

```
kick cornell.ass -set curves.mode thick
```

Get kick to abort the render if no valid license is found, rather than rendering with a watermark:

```
kick cornell.ass -set options.abort_on_license_fail true
```

Override several parameters:

```
kick first.ass -set options.AA_samples 3 -set options.bucket_size 16
```

### Interactive Mode

To render in interactive mode use the "-ipr q" option. This will allow you to (very crudely) navigate the scene and switch to various debug shading modes such as flat/smooth normals, UVs, wireframe, etc.:

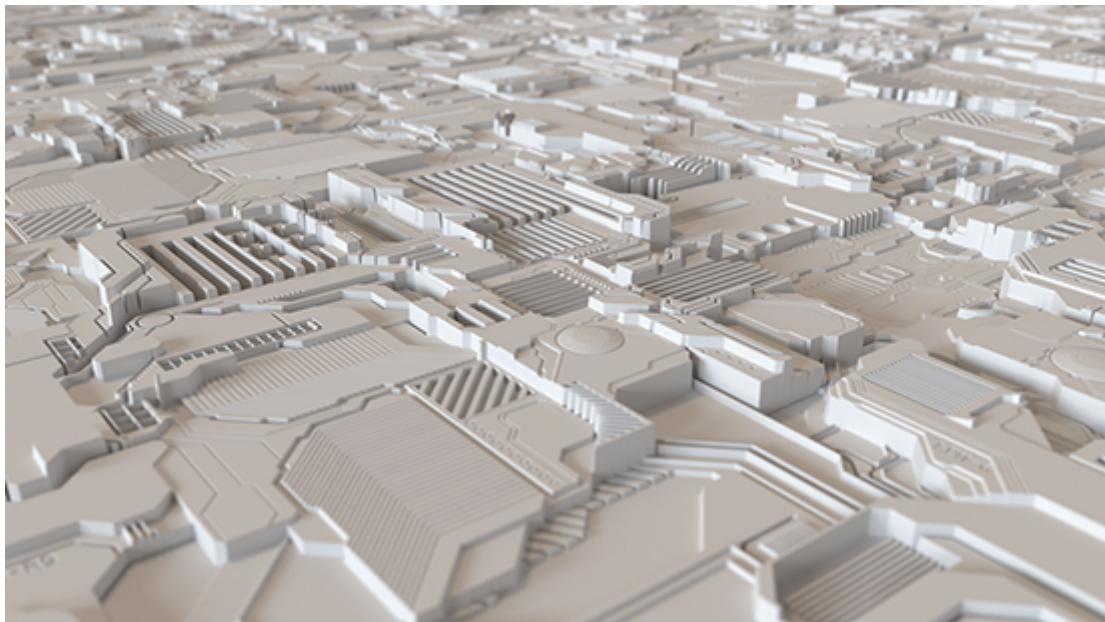
```
kick cornell.ass -ipr q
```

- Navigate (pan/zoom) with the chosen interaction mode: "q" for Quake controls (WASD), and "m" for Maya controls (Alt + Mouse).
- Increase or decrease the image exposure with the "[" and "]" keys.
- Click in the render window and press lowercase "i" to ignore any existing shaders. You can restore the scene shaders by pressing uppercase "i".
- **macOS and Linux:** use the number keys (0,1,2,3,4,5,6,7,8,9) to switch between the various debug shading modes.



# Displacement

## synopsis



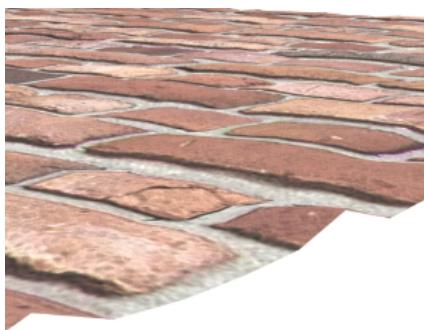
Displacement texture map from JSplacement (rollover image)

Displacement maps can be an excellent tool for adding surface detail that would take far too long using regular modeling methods. Displacement mapping differs from bump mapping in that it alters the geometry, and therefore will have a correct silhouette, and self-shadowing effects. Depending on the type of input, the displacement can occur in two ways: Float, RGB & RGBA inputs will displace along the normal while a vector input will displace along the vector.

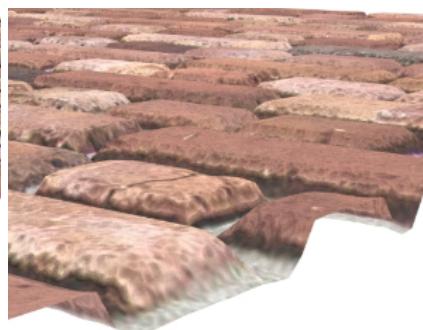
The example above shows how a simple plane, with the addition of a displacement map, can produce an interesting looking simple scene.

You should ensure that your base mesh geometry has a sufficient number of polygons otherwise subtle differences can occur between the displaced low-resolution geometry and the high-resolution mesh from which it was generated.

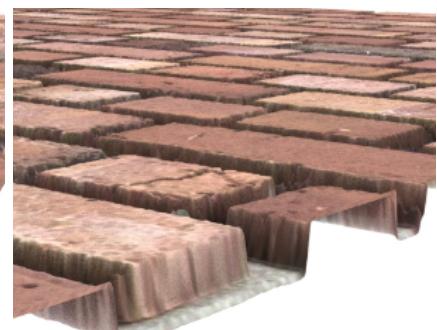
Make sure that you use a 32-bit or 16-bit floating-point format to store your image, and not an integer format. An integer format will not work correctly. This is because integer formats do not support negative pixel values, which are used by floating-point displacement maps.



Subdivision Iterations: 2



Subdivision Iterations: 4



Subdivision Iterations: 8

Changing the Subdivision Type to either Catclark or Linear subdivision rules and increasing the iterations will improve the displacement quality. In this example, the Subdivision Iterations have been increased to 8.

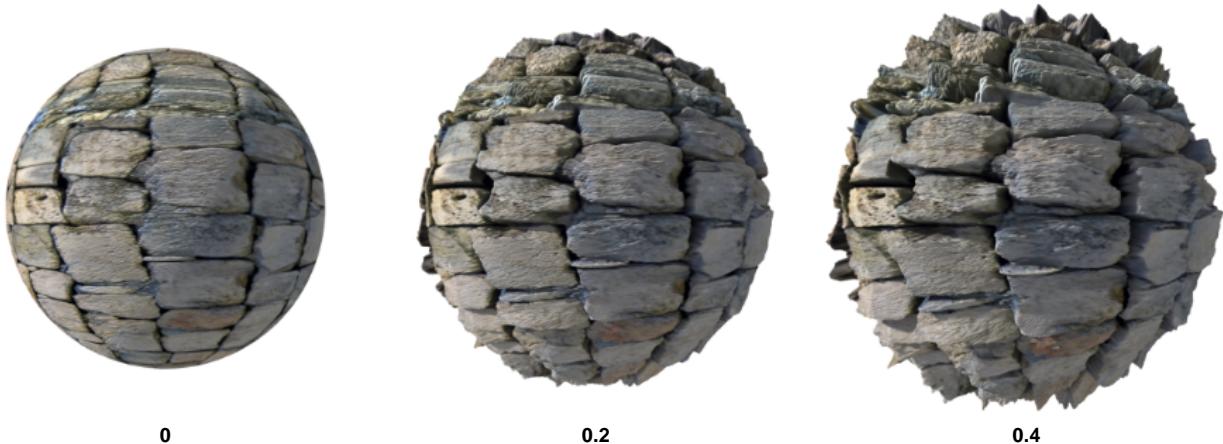
You must be careful when increasing the number of subdivision iterations (each iteration quadruples the geometry). This subdivision happens at render time, whenever a ray hits the bound box of the object. This is a better choice compared to increasing the subdivisions of the mesh within the DCC software (which will send the tessellated geometry to the renderer).

### **`arnold_displacement_settings`**

It is possible to set displacement settings on a per-face or per-object basis. However, any values that are entered in the Arnold attributes of the displacement node will override those settings. With multiple displacement shaders per object, and since an object can only have one value of Bounds Padding, Arnold takes the maximum value from all of them. Autobump poses the same problem as with padding, and so Arnold enables it if at least one of the displacement shaders has it enabled.

### **`height`**

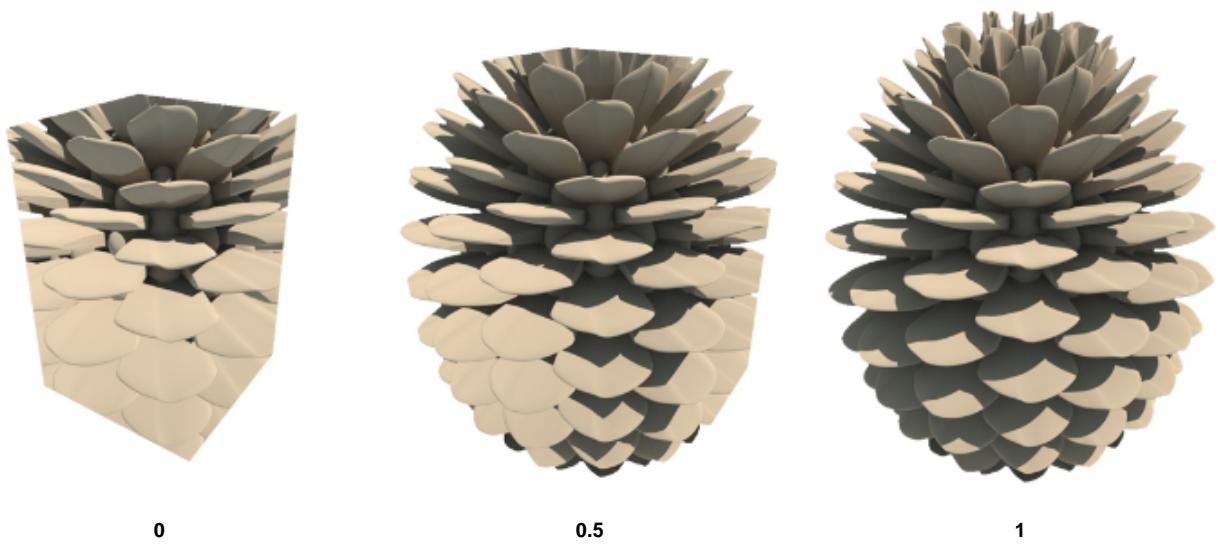
Controls the amount of displacement. Displacement height can have either positive or negative values. This attribute only applies with normal displacement. You can use this value to compensate for any inconsistencies between the exported displacement map and the low-resolution geometry.



### **`bounds_padding`**

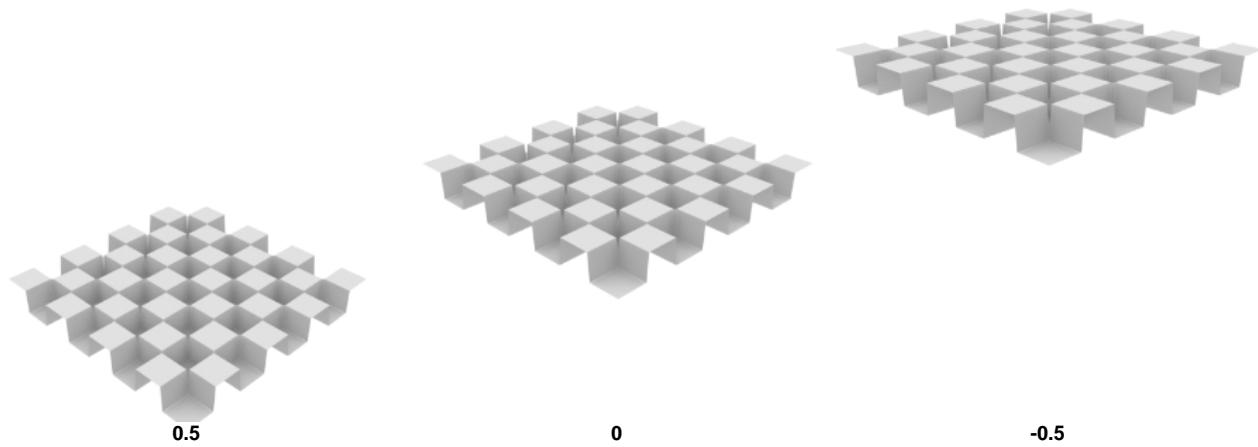
Padding defines how much to extend the bounding box of the object so that it can include any additional displacement coming from the displacement shader. When the bounding box is hit first by a ray, the displacement will be computed, so an unnecessarily high value will decrease the rendering efficiency. On the other hand, a low value could result in a clipping of the displaced mesh.

The proper workflow for displacement in Arnold is to have the shader give the final displacement value and then to offset the bounding box with the `bounds_padding` attribute.



### **zero\_value**

This is a floating point value which is applied as a shift to the displacement amount. It defines the value of the displacement map that is considered to be zero displacement. This value can vary depending on how the displacement map has been generated.



### **autobump**

Autobump puts the high frequencies of a displacement map into the bump attribute so that you do not need as many Subdivision Iteration values. It is enabled by default in the Arnold attributes of the displacement node.

The autobump algorithm needs UV coordinates to compute surface tangents. Make sure your polymesh has a UV set applied.

#### **Technical information:**

When autobump is enabled, Arnold makes a copy of all of the vertices of a mesh before displacement (let's call that the "reference" mesh, or Pref). Prior to shading at some surface point on the displaced surface P, the equivalent Pref for that point is found on the non-displaced surface, and the displacement shader is evaluated there (at Pref) to estimate what would be the equivalent normal at P if we had subdivided the polymesh at an insanely high tessellation rate.

The main difference between Arnold's autobump and using the displacement shader for bump mapping is that autobump has access to Pref

whereas bump2d does not and would be executing the displacement shader on already-displaced points which could "compound" the displacement amounts.

The only extra storage is for copying P prior to displacement. There is no analysis of the displacement map; Arnold displaces vertices purely based on where they "land" in the displacement map (or procedural) regardless of whether it happens to "hit" a high-frequency spike or not.



Disabled (subdivision iterations 2)

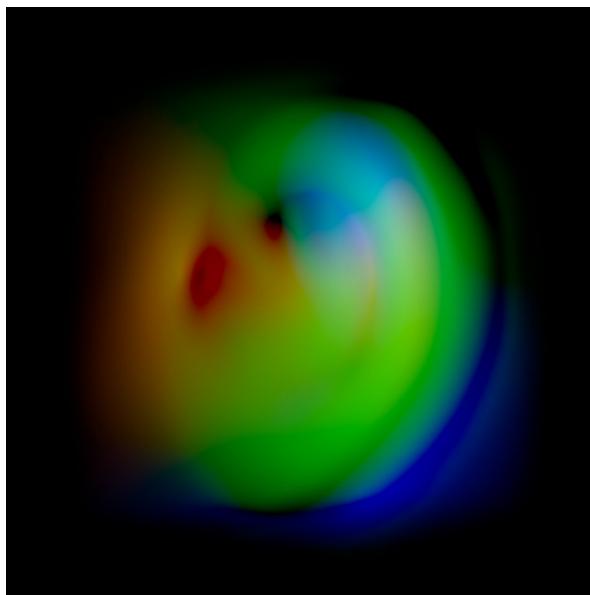


Disabled (subdivision iterations 5)



Enabled (subdivision iterations 2)

### vector\_displacement



Ear tangent space vector displacement map from Mudbox

Polygon plane displaced with ear vector displacement map

Traditional displacement maps are not used for any surface change that is not perpendicular to the base mesh's polygons. Vector displacement maps can displace in directions other than the face normal, which is much more flexible. Vector displacement uses the color channels that specify a vector in a certain space to displace the vertices of the geometry in that direction and magnitude.

# Plugins

Plugins are used to extend Arnold with custom node types. Custom shaders, cameras, filters, drivers, procedural geometry, volumes, implicit shapes and color managers can be created.

- [Creating a Simple Plugin](#)
- [Multiple Nodes in a Library](#)
- [Metadata Files](#)
- [Camera Nodes](#)
- [Procedural Nodes](#)
  - [Random Flake Procedural](#)
  - [Implementing a simple instancer procedural](#)
  - [Large Datasets from Procedurals](#)
- [Filter Nodes](#)
- [Driver Nodes](#)
  - [Simple Driver](#)
  - [Display Driver](#)
  - [Raw Driver](#)

# Creating a Simple Plugin

## *Prerequisites*

To be able to create Arnold shaders, you need a C++ compiler and the Arnold SDK.

Any recent C++ compiler may be used to compile plugins. Typically GCC or Clang on Linux, Xcode (Clang) on macOS, or Visual Studio on Windows.

You will need to download and extract the Arnold SDK to the folder of your choice. For the purposes of this tutorial, we will assume that you will set the environment variable `ARNOLD_PATH` to the path where the Arnold SDK is located.

## *Simple Shader Example*

Here's a very simple example shader plugin, contain a single shader node named `simple` that takes a single `color` parameter and outputs it.

## simple\_shader.cpp

[Expand source](#)

```
#include <ai.h>

AI_SHADER_NODE_EXPORT_METHODS(SimpleMethods);

enum SimpleParams { p_color };

node_parameters
{
    AiParameterRGB("color", 0.7f, 0.7f, 0.7f);
}

node_initialize
{
}

node_update
{
}

node_finish
{
}

shader_evaluate
{
    sg->out.RGB() = AiShaderEvalParamRGB(p_color);
}

node_loader
{
    if (i > 0)
        return false;
    node->methods      = SimpleMethods;
    node->output_type  = AI_TYPE_RGB;
    node->name         = "simple";
    node->node_type    = AI_NODE_SHADER;
    strcpy(node->version, AI_VERSION);
    return true;
}
```

### Short Description

**node\_initialize** is called only once for each instance of the shader, at the beginning of the first render.

**node\_update** is called once per render call (so, multiple times during progressive rendering or IPR) for each instance of the shader. And this includes the first render.

Summing up, for N calls to **AiRender()**, you have one call to **node\_initialize**, then N calls to **node\_update**. These numbers are not related to the number of threads at all.

**node\_finish** is called once for each instance of the shader when an **AiEnd()** is called.

### Compiling

Here are example commands for compiling from the command line.

## Linux

```
export ARNOLD_PATH=/path/to/arnold
c++ simple_shader.cpp -o simple_shader.so -Wall -O2 -shared -fPIC
-I$ARNOLD_PATH/include -L$ARNOLD_PATH/bin -lai
```

## macOS

```
export ARNOLD_PATH=/path/to/arnold
c++ simple_shader.cpp -o simple_shader.dylib -Wall -O2 -shared -fPIC
-I$ARNOLD_PATH/include -L$ARNOLD_PATH/bin -lai
```

## Windows Visual Studio command prompt

```
set ARNOLD_PATH=c:/path/to/arnold
cl /LD simple_shader.cpp /I %ARNOLD_PATH%/include %ARNOLD_PATH%/lib/ai.lib /link
/out:simple_shader.dll
```

Compiling in an IDE such as Visual Studio or Xcode is also possible. The main steps are:

- Create a shared library project.
- Add the Arnold include/ directory to the include directories.
- Add the Arnold bin/ (Linux and macOS) or lib/ (Windows) directory to the library directories.
- Link to libai

## Loading the Plugin

All shared libraries in specified plugin paths that contain a node\_loader will be loaded. Plugins paths may be specified in a few ways.

- options.plugin\_searchpath can contain one or more paths to load plugins from
- The ARNOLD\_PLUGIN\_PATH environment variable can also contain one or more plugin paths
- kick -l <path\_to\_plugin> may be used to load plugins from a specified path
- kick loads plugins from the current working directory

Kick can be used to inspect if the shader is loaded correctly:

```
$ kick -l <path_to_plugin> -info simple
node:      simple
type:      shader
output:    RGB
parameters: 2
filename:   ./simple_shader.so
version:   5.0.0.0
Type      Name          Default
-----  -----
RGB      color          0.7, 0.7, 0.7
STRING   name
```

## **Testing the Plugin**

To test how the shader is working, you can use the following simple scene:

### **simple\_scene.ass**

 [Expand source](#)

```
### Scene using "simple" shader

options
{
    name options
    camera "camera"
}

persp_camera
{
    name camera
    matrix
        1 0 0 0
        0 1 0 0
        0 0 1 0
        0 0 30 1
}

sphere
{
    name pSphereShape2
    center 0 0 0
    radius 10
    shader "simpleShader"
}

simple
{
    name simpleShader
    color 1 0 0
}
```

Render this scene using this command:

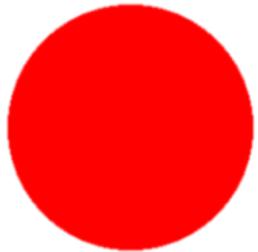
### **Linux or macOS**

```
$ARNOLD_PATH/bin/kick -l <path_to_plugin> simple_scene.ass
```

### **Windows**

```
%ARNOLD%\\bin\\kick -l <path_to_plugin> simple_scene.ass
```

And you will get the following image:

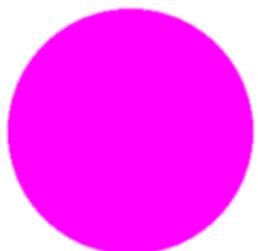


*Figure 1: simpleScene*

If the shader cannot be loaded, you will get this warning:

```
00:00:00 10MB WARNING | [ass] line 28: node "simple" is not installed
```

And this image:



*Figure 2: simpleScene Error*

### **Metadata**

To expose nodes in applications that integrate Arnold, application specific [metadata](#) must be provided.

## Multiple Nodes in a Library

If you create several custom nodes, it can be useful to create a loader that will load all of them from the same library ie. It is also possible to create a metadata file with all the metadata information from all the shaders.

### Two Simple Shaders

The first thing you need are the shaders you want to include; here are two very basic ones. Each node is defined in its own source code file, and both export their methods using the `AI_SHADER_NODE_EXPORT_METHODS` macro.

#### shader1.cpp

```
#include <ai.h>

AI_SHADER_NODE_EXPORT_METHODS(Shader1Mtd);
enum MyShader1Params { p_color };

node_parameters
{
    AiParameterRGB("color1", 0.0f, 0.0f, 0.0f);
}

node_initialize {}
node_update {}
node_finish {}

shader_evaluate
{
    sg->out.RGB() = AiShaderEvalParamRGB(p_color);
}
```

#### shader2.cpp

```
#include <ai.h>

AI_SHADER_NODE_EXPORT_METHODS(Shader2Mtd);
enum MyShader2Params { p_color };

node_parameters
{
    AiParameterRGB("color2", 1.0f, 1.0f, 1.0f);
}

node_initialize {}
node_update {}
node_finish {}

shader_evaluate
{
    sg->out.RGB() = AiShaderEvalParamRGB(p_color);
}
```

### The Loader

When Arnold loads the shared, it will call the `node_loader` entry point several times, each time increasing the index `i`, until one of the calls returns `false`. So for loading two shaders, `node_loader` should return `true` for values of `i` between 0 and 1 and set the corresponding shader data in `node`, and return `false` for all other values.

### loader.cpp

```
#include <ai.h>
#include <cstring>

extern AtNodeMethods* Shader1Mtd;
extern AtNodeMethods* Shader2Mtd;

enum{
    SHADER_1 = 0,
    SHADER_2
};

node_loader
{
    switch (i)
    {
        case SHADER_1:
            node->methods = Shader1Mtd;
            node->output_type = AI_TYPE_RGB;
            node->name = "shader1";
            node->node_type = AI_NODE_SHADER;
            break;

        case SHADER_2:
            node->methods = Shader2Mtd;
            node->output_type = AI_TYPE_RGB;
            node->name = "shader2";
            node->node_type = AI_NODE_SHADER;
            break;

        default:
            return false;
    }

    strcpy(node->version, AI_VERSION);
    return true;
}
```

### Compiling

The commands to compile are similar to a single shader, just using more files.

#### Linux

```
export ARNOLD_PATH=/path/to/arnold
c++ shader1.cpp shader2.cpp loader.cpp -o simple_shaders.so -Wall -O2 -shared -fPIC
-I$ARNOLD_PATH/include -L$ARNOLD_PATH/bin -lai
```

### macOS

```
export ARNOLD_PATH=/path/to/arnold  
c++ shader1.cpp shader2.cpp loader.cpp -o simple_shaders.dylib -Wall -O2 -shared  
-fPIC -I$ARNOLD_PATH/include -L$ARNOLD_PATH/bin -lai
```

### Windows Visual Studio command prompt

```
set ARNOLD_PATH=c:/path/to/arnold  
cl /LD shader1.cpp shader2.cpp loader.cpp /I %ARNOLD_PATH%/include  
%ARNOLD_PATH%/lib/ai.lib /link /out:simple_shaders.dll
```

## Metadata Files

Arnold includes support for specification of nodes and attributes metadata using separate text files.

Each library implementing Arnold nodes could have an associated metadata file, with the same name of the library and the extension ".mtd". For example, a library called `mtoa_shaders.dll` would have a corresponding `mtoa_shaders.mtd`. This metadata files will be automatically loaded when the library is first loaded. As a special case, for built-in Arnold nodes, the name of the metadata file is `arnold.mtd`, as this library has different names on different platforms.

Also, there could be additional metadata files, which can be loaded on demand, by using the API function provided in `ai_metadata.h` (or `ai_metadata.py` for Python):

```
AI_API bool AiMetaDataLoadFile(const char* filename);
```

This file could be used for example to override values or add additional metadata to built-in nodes, such as MtoA and SltoA do for Arnold built-ins.

Finally, third party plugin developers could include a metadata file with their library, in a file with the same name as the library, but ".mtd" extension. In this metadata file, they could include information for all plugins (SltoA, MtoA for now), so that the same file would work for all of them.

### File Format

The format of the metadata files is like this:

```
# Arnold test metaData file
[node wireframe]
desc STRING "This shader can be used to display the edge structure of a geometry."
[attr line_width]    min INT -3
                    max INT 100
                    softmin INT 0
                    softmax INT 1
                    desc STRING "Sets width of the lines used to display edges."
[attr fill_color]   desc STRING "Color used for the faces."
[attr line_color]   desc STRING "Color used for the lines used to display edges."
[attr raster_space] desc STRING "Uses screen space to measure line width."
[attr edge_type]    desc STRING "Selects base geometry element for edge display."
[attr example]      maya.test1 FLOAT -0.63
                    maya.test2 BOOL true
                    maya.test3 BOOL 0
                    maya.test4 RGB 0 1 0
                    maya.test5 POINT2 -1 489
                    maya.test6 INT 0x09ABCDEF
```

There could be any number of nodes in the file. There can also be multiple node sections for the same node (metadata will be applied in the order they are found in the file).

As a convention, we are using metadata names starting with "maya." for Maya specific metadata, and "xsi." for XSI specific metadata. There are some other conventions for standard metadata elements that would be published on a separate document.

### Maya

For MtoA specific metadata, see [Creating a Shader for Maya](#).

## Camera Nodes

Here is the source code for a simple perspective camera node with custom distortion and vignetting :

### mycamera.cpp

```
#include <ai.h>
#include <string.h>

AI_CAMERA_NODE_EXPORT_METHODS(MyCameraMethods)

enum
{
    p_fov
};

struct MyCameraData
{
    float tan_fov;
};

node_parameters
{
    AiParameterFlt("fov", 60.0f);
}

node_initialize
{
    AiCameraInitialize(node);
    AiNodeSetLocalData(node, new MyCameraData());
}

node_update
{
    MyCameraData* data = (MyCameraData*)AiNodeGetLocalData(node);
    data->tan_fov = tanf(AiNodeGetFlt(node, "fov") * AI_DTOR / 2);
    AiCameraUpdate(node, false);
}

node_finish
{
    MyCameraData* data = (MyCameraData*)AiNodeGetLocalData(node);
    delete data;
}

camera_create_ray
{
    const MyCameraData* data = (MyCameraData*)AiNodeGetLocalData(node);
    const AtVector p(input.sx * data->tan_fov, input.sy * data->tan_fov, 1);

    // warp ray origin with a noise vector
    AtVector noise_point(input.sx, input.sy, 0.5f);
    noise_point *= 5;
    AtVector noise_vector = AivNoise3(noise_point, 1, 0.f, 1.92f);
    output.origin = noise_vector * 0.04f;
    output.dir = Aiv3Normalize(p - output.origin);
```

```

// vignetting
const float dist2 = input.sx * input.sx + input.sy * input.sy;
output.weight = 1 - dist2;

// now looking down -Z
output.dir.z *= -1;
}

camera_reverse_ray
{
    const MyCameraData* data = (MyCameraData*)AiNodeGetLocalData(node);

    // Note: we ignore distortion to compute the screen projection
    // compute projection factor: avoid divide by zero and flips when crossing the
    camera plane
    float coeff = 1 / AiMax(fabsf(Po.z * data->tan_fov), 1e-3f);
    Ps.x = Po.x * coeff;
    Ps.y = Po.y * coeff;
    return true;
}

node_loader
{
    if (i != 0) return false;
    node->methods      = MyCameraMethods;
    node->output_type  = AI_TYPE_UNDEFINED;
    node->name         = "mycamera";
    node->node_type    = AI_NODE_CAMERA;
}

```

```

    strcpy(node->version, AI_VERSION);
    return true;
}

```

In `camera_create_ray` you can also compute the direction and position derivatives. This is important for correct filtering. However, you can let Arnold compute the differentials automatically if you leave these fields set to the default 0.0 value.

The following example shows how to compute them in the camera node for a perspective camera. In this case the ray's origin is always the same, but the direction changes from pixel to pixel. Follows some sample code to compute the derivatives for a perspective camera (we do not take into account uv noise):

```

float fov = data->fov;
fov *= (float) (AI_DTOR * 0.5);
float tan_fov = tanf(fov);

...
// scale derivatives
float dsx = input.dsx * tan_fov;
float dsy = input.dsdy * tan_fov;
...

AtVector d = p; // direction vector == point on the image plane
double d_dot_d = AiV3Dot(d, d);
double temp = 1.0 / sqrt(d_dot_d * d_dot_d * d_dot_d);

// already initialized to 0's, only compute the non zero coordinates
output.dDdx.x = (d_dot_d * dsx - (d.x * dsx) * d.x) * temp;
output.dDdx.y = (                - (d.x * dsx) * d.y) * temp;
output.dDdx.z = (                - (d.x * dsx) * d.z) * temp;
output.dDdy.x = (                - (d.y * dsy) * d.x) * temp;
output.dDdy.y = (d_dot_d * dsy - (d.y * dsy) * d.y) * temp;
output.dDdy.z = (                - (d.y * dsy) * d.z) * temp;

// output.dOd* is also initialized to 0s, the correct value

```

Example scene and render:

### mycamera.ass

[Expand source](#)

```

options
{
    xres 1024
    yres 1024
    AA_samples 6
    GI_diffuse_depth 4
    GI_specular_depth 4
}

mycamera
{
    name mycamera
    position 0 1 4
    look_at 0 -0.2 0
}

```

```
    up 0 1 0
}

skydome_light
{
    name myskydome
    intensity 1
    color 1 1 1
    camera 0.0
}

standard_surface
{
    name mystd
    base_color 0.4 0.8 0.4
}

sphere
{
    shader mystd
    matrix
        1 0 0 0
        0 1 0 0
        0 0 1 0
        -1.5 0 0 0
}

sphere
{
    shader mystd
    matrix
        1 0 0 0
        0 1 0 0
        0 0 1 0
        0 0 0 0
}

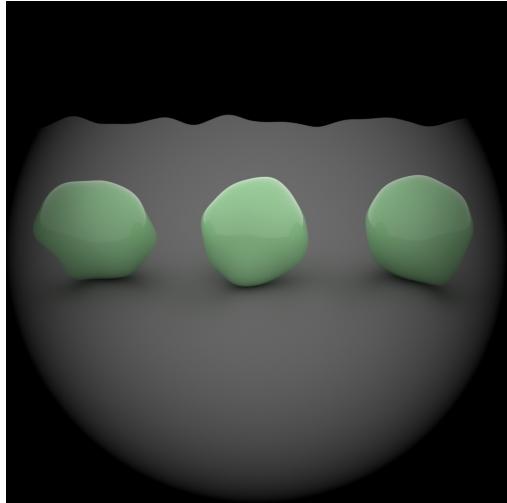
sphere
{
    shader mystd
    matrix
        1 0 0 0
        0 1 0 0
        0 0 1 0
        1.5 0 0 0
}

standard_surface
{
    name myfloor
    base_color 0.2 0.2 0.2
    specular 0
}

plane
{
    name myplane
```

normal 0 1 0

```
point 0 -0.5 0
shader myfloor
}
```



## Procedural Nodes

- Random Flake Procedural
- Implementing a simple instancer procedural
- Large Datasets from Procedurals

## Random Flake Procedural

This is an example of making a geometry-generating procedural for Arnold. There are 2 files in this example; the code for the procedural, and an .ass scene file.

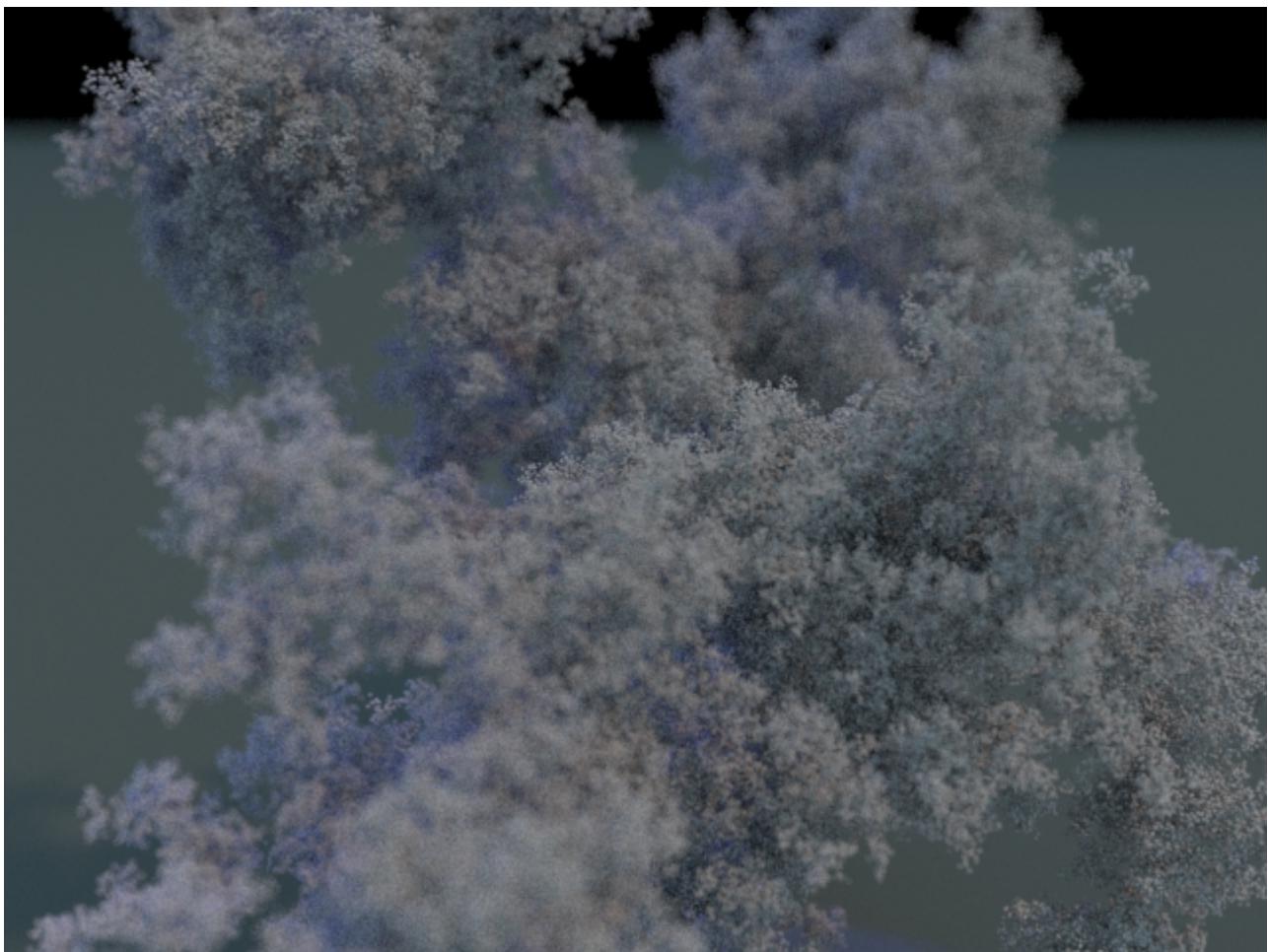
This procedural makes a random pointcloud with fractal distribution in space. There are 5 controls for the generation of the cloud:

- count: the number of points per recursion to generate
- recursions: the number of recursive steps to descend
- flake\_radius: the radius of the little spheres that make up the cloud
- cloud\_radius: the overall radius of the pointcloud
- seed: a random seed to generate a unique random cloud

The algorithm works like this: given a spherical region of radius R, scatter N points into it. For each point, scatter N random points within a radius of 1/2 the previous R, and so on, for X number of recursions, then hang a little sphere of radius S on the resulting points.

Warning: this algorithm creates poorly bound geometry, and is likely to build a few points outside the bounding box.

This is a sample render of a pointcloud with 2,015,539 points:



Here's the procedural code:

```
#include <ai.h>
#include <cstring>
#include <cstdlib>

// Procedural parameters
struct RandomFlake
{
```

```

    int    count;
    float flake_radius;
    float sphere_radius;
    int    recursions;
    int    counter;
    int    num_points;
};

// returns a random vector in a unit sphere with a
// power function to bias it towards the center
static AtVector random_vector(float power)
{
    AtVector out(drand48() - 0.5, drand48() - 0.5, drand48() - 0.5);
    return AiV3Normalize(out) * pow(drand48(), power);
}

// recursive function that creates random flake with fractal clumping
static void make_cloud(RandomFlake *flake, AtArray *point_array, AtArray
*radius_array, AtVector center, float radius, int recursions)
{
    for (int i = 0; i < flake->count; i++)
    {
        AtVector new_center = random_vector(0.5) * radius;
        AiArraySetVec(point_array, flake->counter, new_center + center);
        AiArraySetFlt(radius_array, flake->counter, flake->sphere_radius);
        flake->counter++;
        if (recursions > 1)
            make_cloud(flake, point_array, radius_array, new_center + center, radius *
0.5, recursions - 1);
    }
}

AI_PROCEDURAL_NODE_EXPORT_METHODS(RandomFlakeMtd);

node_parameters
{
    AiParameterInt("count"           , 10);
    AiParameterInt("recursions"     , 5);
    AiParameterFlt("sphere_radius"  , 0.01f);
    AiParameterFlt("flake_radius"   , 10.0f);
    AiParameterInt("seed"           , 0);
}

procedural_init
{
    RandomFlake *flake = new RandomFlake();
    *user_ptr = flake;

    srand48(AiNodeGetInt(node, "seed"));

    flake->count      = AiNodeGetInt(node, "count");
    flake->sphere_radius = AiNodeGetFlt(node, "sphere_radius");
    flake->flake_radius = AiNodeGetFlt(node, "flake_radius") -
flake->sphere_radius;
    flake->recursions = AiNodeGetInt(node, "recursions");
    flake->counter    = 0;
}

```

```

flake->num_points = 0;
for (int i = 0; i < flake->recursions; i++)
    flake->num_points += pow(flake->count, i);
AiMsgInfo("[random_flake] number of points: %d", flake->num_points);

return true;
}

procedural_cleanup
{
    RandomFlake *flake = (RandomFlake*)user_ptr;
    delete flake;
    return true;
}

procedural_num_nodes
{
    return 1;
}

procedural_get_node
{
    RandomFlake *flake = (RandomFlake*)user_ptr;
    AtArray *point_array      = AiArrayAllocate(flake->num_points, 1, AI_TYPE_VECTOR);
    AtArray *radius_array     = AiArrayAllocate(flake->num_points, 1, AI_TYPE_FLOAT);

    make_cloud(flake, point_array, radius_array, AI_V3_ZERO, flake->flake_radius,
    flake->recursions - 1);

    // create node with procedural node as parent
    AtNode *points_node = AiNode("points", "flake", node);
    AiNodeSetArray(points_node, "points", point_array);
    AiNodeSetArray(points_node, "radius", radius_array);
    AiNodeSetStr  (points_node, "mode"   , "sphere");

    return points_node;
}

node_loader
{
    if (i>0)
        return false;

    node->methods      = RandomFlakeMtd;
    node->output_type  = AI_TYPE_NONE;
    node->name         = "random_flake";
    node->node_type    = AI_NODE_SHAPE_PROCEDURAL;
    strcpy(node->version, AI_VERSION);
}

```

```
    return true;  
}
```

This is the .ass file that renders the above image:

```
options  
{  
    AA_samples 6  
    outputs "RGB RGB myfilter mydriver"  
    xres 640  
    yres 480  
    GI_diffuse_depth 1  
}  
  
driver_jpeg  
{  
    name mydriver  
    filename "randflake.jpg"  
}  
  
gaussian_filter  
{  
    name myfilter  
}  
  
plane  
{  
    name myplane  
    point 0 -10 0  
    normal 0 1 0  
    shader planeshader  
}  
  
random_flake  
{  
    name myrandflake  
    shader flakeshader  
    count 6  
    recursions 9  
    sphere_radius 0.015  
    flake_radius 10  
    seed 4  
}  
  
lambert  
{  
    name flakeshader  
    Kd 0.7  
}  
  
lambert  
{  
    name planeshader
```

```
Kd_color .15 .2 .2
}

persp_camera
{
    name mycamera
    focus_distance 11
    aperture_size .15
    position -5 5 15
    look_at 0 0 0
}

point_light
{
    name key
    position 100 200 100
    radius 4
    color 1 0.6 0.4
}

spot_light
{
    name kicker
    position -200 200 -500
    look_at 0 0 0
    cone_angle 2
    color .1 0.2 3
}

skydome_light
{
    name mysky
```

```
    color 0.6 0.85 1  
    intensity .5  
}
```

See [Creating a Simple Plugin](#) for compilation commands.

## Implementing a simple instancer procedural

Here is a simple instancer procedural developed based on code by Gael Honorez.

This should be a good starting point for learning how to write an Arnold procedural, since it covers reading user declared procedural parameters, how to create a new node, set its parameters and setup motion blur matrices.

```
#include <ai.h>
#include <sstream>
#include <vector>

struct Instancer
{
    std::string name;
    AtArray *path_indices;
    AtArray *path_start_indices;
    AtArray *instance_matrix;
    std::vector<AtNode*> objects;
    int num_instances;
};

AI_PROCEDURAL_NODE_EXPORT_METHODS(InstancerMtd);

node_parameters
{
    AiParameterArray("path_indices" , AiArray(0, 1, AI_TYPE_INT));
    AiParameterArray("path_start_indices" , AiArray(0, 1, AI_TYPE_INT));
    AiParameterArray("objects" , AiArray(0, 1, AI_TYPE_STRING));
    AiParameterArray("instance_matrix" , AiArray(0, 1, AI_TYPE_MATRIX));
}

procedural_init
{
    Instancer *instancer = new Instancer();
    *user_ptr = instancer;

    // get procedural name so it's the prefix of instanced nodes' name
    instancer->name = AiNodeGetStr(node, "name");
    instancer->path_indices = AiNodeGetArray(node, "path_indices");
    instancer->num_instances = AiArrayGetNumElements(instancer->path_indices);

    // this is the start index of each instance
    instancer->path_start_indices = AiNodeGetArray(node, "path_start_indices");
    AiMsgInfo("[instancer] number of instances: %d",
    AiArrayGetNumElements(instancer->path_indices));

    AtArray *object_names = AiNodeGetArray(node, "objects"); // to get the AtNode
    pointers for all the master nodes

    // allocate and create AtNode pointer array
    AiMsgInfo("[instancer] getting master node list ...");

    instancer->objects.resize(AiArrayGetNumElements(object_names));
    for (unsigned int i = 0; i < AiArrayGetNumElements(object_names); i++)
    {
        const char *master_name = AiArrayGetStr(object_names, i);
        AiMsgInfo("[instancer] adding node %s to master list", master_name);
    }
}
```

```

AtNode *node = AiNodeLookUpByName(master_name);
instancer->objects[i] = node;
if (node == NULL)
    AiMsgWarning("[instancer] node %s does not exist!", master_name);
}

// get matrices pointer and number of motion keys
AtArray *instance_matrix = AiNodeGetArray(node, "instance_matrix");
instancer->instance_matrix = instance_matrix;
AiMsgInfo("[instancer] instance_matrix elements: %d, motion keys: %d",
AiArrayGetNumElements(instance_matrix), AiArrayGetNumKeys(instance_matrix));

return true;
}

procedural_cleanup
{
    Instancer *instancer = (Instancer*)user_ptr;
    delete instancer;
    return true;
}

procedural_num_nodes
{
    // return how many nodes to generate
    Instancer *instancer = (Instancer*)user_ptr;
    return AiArrayGetNumElements(instancer->instance_matrix);
}

procedural_get_node
{
    Instancer *instancer = (Instancer*)user_ptr;

    int instance_index = AiArrayGetInt(instancer->path_start_indices, i); // get
particle index for this instance
    int path_index = AiArrayGetInt(instancer->path_indices, instance_index);
    AtNode *object = instancer->objects[path_index];

    AtNode *currentInstance = AiNode("ginstance", "instancer", node); // initialize
node as child of procedural node

    std::stringstream numStr; // setup node name by concatenate procedural name with
instance node number
    numStr << "_" << instance_index;
    std::string currentName(instancer->name + numStr.str());

    AiNodeSetStr(currentInstance, "name", currentName.c_str());
    AiNodeSetPtr(currentInstance, "node", (void *)object); // setup ginstance node
parameter
    AiNodeSetBool(currentInstance, "inherit_xform", false); // usually instancer
doesn't move around, but maybe this should be set to true?

    int num_motion_keys = AiArrayGetNumKeys(instancer->instance_matrix);
    if (num_motion_keys > 1)
    {
        // allocate and assign matrices

```

```
AtArray *matrices = AiArrayAllocate(1, num_motion_keys, AI_TYPE_MATRIX);
for (int j = 0; j < num_motion_keys; j++)
{
    AtMatrix matrix = AiArrayGetMtx(instancer->instance_matrix, instance_index
+ instancer->num_instances * j);
    AiArraySetMtx(matrices, j, matrix);
}
AiNodeSetArray(currentInstance, "matrix", matrices);
}
else
{
    AtMatrix matrix = AiArrayGetMtx(instancer->instance_matrix, instance_index);
    AiNodeSetMatrix(currentInstance, "matrix", matrix);
}

return currentInstance;
}

node_loader
{
    if (i>0)
        return false;
    node->methods      = InstancerMtd;
    node->output_type  = AI_TYPE_NONE;
    node->name         = "instancer";
    node->node_type    = AI_NODE_SHAPE_PROCEDURAL;
    strcpy(node->version, AI_VERSION);
```

```
    return true;  
}
```

## Large Datasets from Procedurals

This tutorial shows how to create as much geometry as possible in a finite amount of RAM in a procedural.

Procedurally generated geometry for rendering in Arnold goes through 3 phases of RAM consumption. The first phase is the generation of data, next is the filling of Arnold's data structures, and last is the ray acceleration data structure creation (the Bounding Volume Hierarchy, or BVH, is usually the largest). When this is all done, the actual rendering begins, which does not use much more RAM. Directly building arrays of data into Arnold's data structures is the most efficient method, if it is possible. Users do not have control over RAM consumption of the BVH.

### The Mandelbulb

In our example, we are generating a "Mandelbulb" 3D version of the Mandelbrot and Julia sets. This algorithm involves iterating a function ( $Z^n+C$ ) and seeing if it exits a sphere of radius 2; if it stays inside for a set number of steps, it is considered a "prisoner point" and a small sphere is put there. Rendering a Mandelbulb as a dense grid of spheres is neither the most elegant nor the most efficient way to display this mathematical entity; we are just using this as a method to create large amounts of data for the purposes of this tutorial.



This [animation](#) shows the sphere size in a closeup on a Mandelbulb made from approximately 1/4 of a billion spheres. It was rendered on a laptop computer in 8 GB of RAM:

### Breaking Generation into Chunks

It is not possible to know how many spheres we will have when we are done, so we cannot fill Arnold's arrays directly; instead, we fill a linked list and then use that to fill the arrays in a second pass. If we fill RAM with a giant linked list of all spheres, we would then need to allocate an array for Arnold to copy the data into; we would only be able to use half of the RAM in a system using this method. Instead, in our example, we break the task into smaller chunks, with each chunk filling a linked list, allocating an array, and then deleting the linked list. This allows us to fill our RAM to the top with renderable geometry. For our example, we simply broke the Mandelbulb up into slabs on the X-axis.

### Multi-Threading

Running the math that generates the points takes CPU power, and many modern systems have access to multiple CPUs on a single system. In order to fill the RAM as efficiently as possible, we take each of our chunks and break it into sub-chunks, allowing a single CPU to fill each of those in parallel, using Arnold's `AiThreadCreate()`. There are numerous other accelerations that can be incorporated into this, such as SIMD sin/cos functions or possibly offloading computations to a GPU or other methods, but we leave these types of optimizations out of our example.

### Source Code

Below is the source code for the procedural:

### mandelbulb.cpp

```
#include <ai.h>
#include <cstring>
#include <cstdlib>
#include <vector>

using namespace std;

// procedural parameters
struct Mandelbulb
{
    int      grid_size;          // sample grid resolution
    int      max_iterations;    // max Z^n+C iterations to try
    float    power;              // exponent, the "n" in Z^n+C
    float    sphere_scale;       // scales the spheres
    float    orbit_threshold;   // clears out a hollow center in the set
    int      chunks;             // number of "chunks" for RAM management
    int      threads;            // number of threads to use
    bool     julia;              // mandelbrot/julia set switch
    AtVector julia_C;           // C value for julia sets (unused for mandelbrot)
    int      counter;
};

// returns Z^n + C
// for more info: http://www.skytopia.com/project/fractal/2mandelbulb.html#formula
// this function is called often and is not very fast,
// this would be an obvious place to add optimizations,
// such as SIMD sin() functions or whatnot
static AtVector iterate(AtVector Z, float n, AtVector C)
{
    AtVector Zsquared;
    float r2 = AiV3Dot(Z, Z);
    float theta = atan2f(sqrtf(Z.x * Z.x + Z.y * Z.y), Z.z);
    float phi = atan2f(Z.y, Z.x);
    float r_n;
    if (n == 8)
        r_n = r2 * r2;
    else
        r_n = powf(r2, n*.5f);
    const float sin_theta_n = sinf(theta * n);

    Zsquared.x = r_n * sin_theta_n * cosf(phi * n);
    Zsquared.y = r_n * sin_theta_n * sinf(phi * n);
    Zsquared.z = r_n * cosf(theta * n);
    return Zsquared + C;
}

AI_PROCEDURAL_NODE_EXPORT_METHODS(MandelbulbMtd);

node_parameters
{
    AiParameterInt("grid_size"      , 1000);
    AiParameterInt("max_iterations" , 10);
```

```

        AiParameterFlt("power"           , 8);
        AiParameterFlt("sphere_scale"   , 1);
        AiParameterFlt("orbit_threshold", 0.05);
        AiParameterInt("chunks"         , 30);
        AiParameterInt("threads"        , 4);
        AiParameterBool("julia"          , false);
        AiParameterVec("julia_C"         , 0, 0, 0);
    }

    // we read the UI parameters into their global vars
procedural_init
{
    Mandelbulb *bulb = new Mandelbulb();
    *user_ptr = bulb;

    bulb->grid_size = AiNodeGetInt(node, "grid_size");
    bulb->max_iterations = AiNodeGetInt(node, "max_iterations");
    bulb->power = AiNodeGetFlt(node, "power");
    bulb->sphere_scale = AiNodeGetFlt(node, "sphere_scale");
    bulb->orbit_threshold = AiSqr(AiNodeGetFlt(node, "orbit_threshold"));
    bulb->chunks = AiNodeGetInt(node, "chunks");
    bulb->threads = AiClamp(AiNodeGetInt(node, "threads"), 1, AI_MAX_THREADS);
    bulb->julia = AiNodeGetBool(node, "julia");
    bulb->julia_C = AiNodeGetVec(node, "julia_C");
    bulb->counter = 0;

    return true;
}

procedural_cleanup
{
    Mandelbulb *bulb = (Mandelbulb*)user_ptr;
    delete bulb;
    return true;
}

// we will create one node per chunk as set in the UI
procedural_num_nodes
{
    Mandelbulb *bulb = (Mandelbulb*)user_ptr;
    return bulb->chunks;
}

// this is the function that gets run on each thread
// the function ( $Z^n+C$ ) is sampled at all points in a regular grid
// prisoner points with an orbit greater than bulb->orbit_threshold are added
// the total grid is broken into bulb->chnks number of slabs on the X axis
// each slab is broken into bulb->threads number of sub slabs
// and the start and end value in X is passed in and that section is sampled
void fillList(Mandelbulb *bulb, int start, int end, int chunknum, vector<AtVector>& list)
{
    float inv_grid_size = 1.0f / bulb->grid_size;
    // these vars are for a crude counter for percent completed
    unsigned int modder = static_cast<unsigned int>(float(end-start) / 5);
    int levelcount = 0;
    int percent = 0;
}

```

```

int localcounter = 0;

// only samples X in the range "start" to "end"
for (int X = start; X < end; X++) {
    levelcount++;
    // echo out some completion info
    if (modder > 0) {
        if ((levelcount%modder) ==0 ) {
            percent += 10;
            AiMsgInfo("[mandelbulb] %d percent of chunk %d", percent, chunknum);
        }
    }

    // samples all points in Y and Z
    for (int Y = 0; Y < bulb->grid_size; Y++) {
        for (int Z = 0; Z < bulb->grid_size; Z++) {
            AtVector sample;
            sample.x = (X * inv_grid_size - 0.5f) * 2.5f;
            sample.y = (Y * inv_grid_size - 0.5f) * 2.5f;
            sample.z = (Z * inv_grid_size - 0.5f) * 2.5f;
            // init the iterator
            AtVector iterator = sample;
            // now iterate the Z^n+C function bulb->max_iterations number of times
            for (int iter = 0; iter < bulb->max_iterations; iter++) {
                if (AiV3Dot(iterator,iterator) > 4)
                    break; //orbit has left the max radius of 2....
                if (bulb->julia) {
                    // each UI value of C creates a full Julia set
                    iterator = iterate(iterator,bulb->power,bulb->julia_C);
                } else {
                    // Mandelbrot set is the centerpoints of all Julia sets
                    iterator = iterate(iterator,bulb->power,sample);
                }
            }

            // tiny orbits are usually inside the set, disallow them.
            bool allowit = AiV3Dist2(sample,iterator) >= bulb->orbit_threshold;

            // if the orbit is inside radius 2
            // and its endpoint travelled greater than orbitthresh
            if (AiV3Dot(iterator, iterator) < 4 && allowit) {
                bulb->counter++; // increment global counter
                localcounter++; // increment local counter
                // this is a prisoner point, add it to the set
                list.push_back(sample);
            }
        }
    }
}

AiMsgInfo("[mandelbulb] finished 1 thread of chunk %d, new total new points %d",
chunknum, localcounter);
}

// this builds the "points" node in Arnold and sets
// the point locations, radius, and sets it to sphere mode
static AtNode *build_node(const AtNode *parent, Mandelbulb *bulb, vector<AtVector>&

```

```

list)
{
    AtArray *pointarr = AiArrayConvert(list.size(), 1, AI_TYPE_VECTOR, &list[0]);
    vector<AtVector>().swap(list); // clear data used by points vector.
    AtNode *currentInstance = AiNode("points", "mandelbulb", parent); // initialize
node as child of procedural
    AiNodeSetArray(currentInstance, "points", pointarr);
    AiNodeSetFlt(currentInstance, "radius", (2.0f/bulb->grid_size) *
bulb->sphere_scale);
    AiNodeSetInt(currentInstance, "mode", 1);
    return currentInstance;
}

// a data structure to hold the arguments for the thread
// corresponds to the arguments to the fillList() function
struct ThreadArgs {
    Mandelbulb *bulb;
    int start;
    int end;
    int i;
    vector<AtVector> list;
};

// a function to be passed for the thread to execute
// basically a wrapper to the fillList() function
unsigned int threadloop(void *pointer)
{
    ThreadArgs *thread_args = (ThreadArgs*) pointer;
    fillList(thread_args->bulb, thread_args->start, thread_args->end,
thread_args->i, thread_args->list);
    return 0;
}

// this is the function that Arnold calls to request the nodes
// that this procedural creates.
procedural_get_node
{
    Mandelbulb *bulb = (Mandelbulb*)user_ptr;

    // determine the start and end point of this chunk
    float chunksize = float(bulb->grid_size) / float(bulb->chunks);
    int start = static_cast<int>(i*chunksize);
    int end = static_cast<int>((i+1)*chunksize);
    if (end>bulb->grid_size)
        end = bulb->grid_size;
    float range = end - start;

    // make an array of arguments for the threads
    vector<ThreadArgs> thread_args(bulb->threads);
    vector<void*> threads(bulb->threads);

    // now loop through and launch the threads
    for (int tnum = 0; tnum < bulb->threads; tnum++) {
        // figure out the threads start and end points for the sub-chunks
        int tstart = start + static_cast<int>((range/bulb->threads)*tnum);
        int tend = start + static_cast<int>((range/bulb->threads)*(tnum+1));
        thread_args[tnum].start = tstart;
    }
}

```

```

    thread_args[tnum].end = tend;
    thread_args[tnum].i = i;
    thread_args[tnum].bulb = bulb;
    threads[tnum] = AiThreadCreate(threadloop, &thread_args[tnum], 0);
}

// using AiThreadWait, wait 'til the threads finish
size_t listlength = 0;
for (int tnum = 0; tnum < bulb->threads; tnum++) {
    AiThreadWait(threads[tnum]);
    // sum up the length of all threads lists
    listlength += thread_args[tnum].list.size();
}

// a vector to hold all the point data
vector<AtVector> allpoints;
allpoints.reserve(listlength);

// concatenate all the vectors returned by the threads
for (int tnum = 0; tnum < bulb->threads; tnum++) {
    allpoints.insert(allpoints.end(), thread_args[tnum].list.begin(),
thread_args[tnum].list.end());
    vector<AtVector>().swap(thread_args[tnum].list); // clear data
}

for (int k = 0; k < bulb->threads; k++)
    AiThreadClose(threads[k]);

AiMsgInfo("[mandelbulb] total sphere count: %d", bulb->counter);

// if it's empty, return a null and Arnold handles it well.
// passing a node with no geometry causes errors.
if (listlength == 0)
    return NULL;

// build the AtNode
return build_node(node, bulb, allpoints);
}

node_loader
{
    if (i>0)
        return false;
    node->methods      = MandelbulbMtd;
    node->output_type  = AI_TYPE_NONE;
    node->name         = "mandelbulb";
    node->node_type    = AI_NODE_SHAPE_PROCEDURAL;
}

```

```
    strcpy(node->version, AI_VERSION);
    return true;
}
```

### Example Scene

The following .ass file generates an image similar to the one above:

```
mandelbulb.ass

options
{
    AA_samples 9
    outputs "RGBA RGBA /out/arnold1:gaussian_filter /out/arnold1:jpeg"
    xres 960
    yres 540
    GI_diffuse_depth 1
    GI_specular_depth 1
    GI_diffuse_samples 3
}

driver_jpeg
{
    name /out/arnold1:jpeg
    filename "mandelbulb.jpg"
}

gaussian_filter
{
    name /out/arnold1:gaussian_filter
}

persp_camera
{
    name /obj/cam1
    fov 54.512329
    focus_distance 1 2 FLOAT 3.0099871 3.01
    aperture_size 0.004
    aperture_blades 5
    matrix 1 2 MATRIX
    0.9011746 0.010277364 0.43333444 0
    0.15803696 0.92311317 -0.35055155 0
    -0.4036195 0.38439101 0.83026195 0
    -1.38442 1.3010246 2.6316857 1

    0.90098524 0.010277364 0.43372801 0
    0.15819006 0.92311317 -0.35048249 0
    -0.40398207 0.38439101 0.83008558 0
    -1.386554 1.3010246 2.6282051 1
    shutter_start 0.25
    shutter_end 0.75
}

skydome_light
{
```

```
name mysky
intensity 1
}

utility
{
    name /shop/utility1
    shade_mode "lambert"
    color 0.5 0.5 0.5
}

plane
{
    name /obj/FLOOR:/shop/utility1:plane_0
    point 0.0 -0.5 0.0
    normal 0.0 1.0 0.0
    shader "/shop/utility1"
}

standard_surface
{
    name /shop/standard1
    base 0.9
    base_color 0.7 0.7 0.7
    specular_roughness 0.167138
}

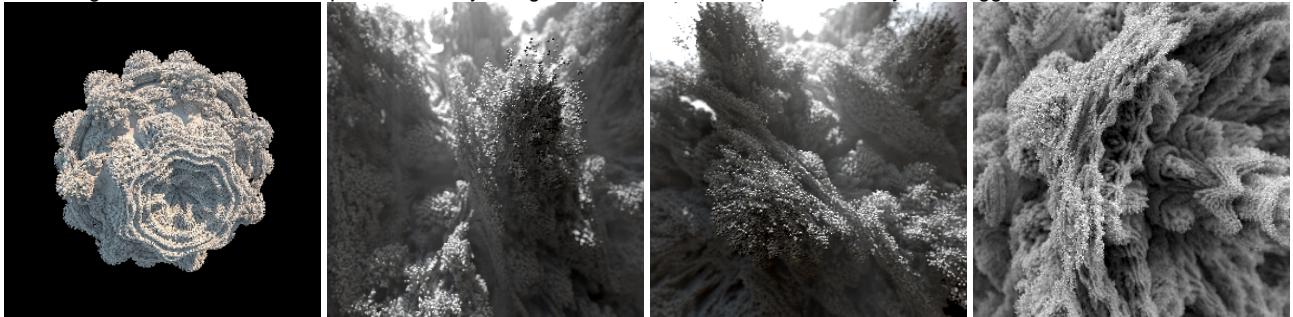
mandelbulb
{
    name /obj/MandelJulia_Procedural1
    shader "/shop/standard1"

    grid_size 1600
    max_iterations 10
    power 8
    sphere_scale 1
    orbit_threshold 0.05
    chunks 30
    threads 16
    julia off
    julia_C -0.161224 1.04 0.183673
}

point_light
{
    name /obj/arnold_light2
    radius 5
    matrix
        1 0 0 0
        0 1 0 0
        0 0 1 0
        -10 10 0 1
    color 1 0.5996 0.076
```

```
intensity 300  
samples 2  
}
```

Here is a high-resolution, 2.5 billion sphere render by Thiago Ize and some close-ups rendered by Lee Griggs.



## Filter Nodes

### *Example*

This is an example filter node plugin, equivalent to the builtin `gaussian_filter`.

#### **custom\_gaussian\_filter.cpp**

```
#include <ai.h>
#include <string.h>

AI_FILTER_NODE_EXPORT_METHODS(CustomGaussianFilterMtd);

struct FilterData
{
    float width;
};

node_parameters
{
    AiParameterFlt("width", 2.0f);
}

node_initialize
{
    AiFilterInitialize(node, false, NULL);
    AiNodeSetLocalData(node, new FilterData());
}

node_update
{
    FilterData* filter_data = (FilterData*)AiNodeGetLocalData(node);
    filter_data->width = AiNodeGetFlt(node, "width");
    AiFilterUpdate(node, filter_data->width);
}

node_finish
{
    FilterData* filter_data = (FilterData*)AiNodeGetLocalData(node);
    delete filter_data;
}

filter_output_type
{
    switch (input_type)
    {
        case AI_TYPE_RGBA:
            return AI_TYPE_RGBA;
        default:
            return AI_TYPE_NONE;
    }
}

filter_pixel
{
    FilterData* filter_data = (FilterData*)AiNodeGetLocalData(node);
```

```

const float width = filter_data->width;

float aweight = 0.0f;
AtRGBA avalue = AI_RGBA_ZERO;

while (AiAOVSampleIteratorGetNext(iterator))
{
    // take into account adaptive sampling
    float inv_density = AiAOVSampleIteratorGetInvDensity(iterator);
    if (inv_density <= 0.f)
        continue;

    // determine distance to filter center
    const AtVector2& offset = AiAOVSampleIteratorGetOffset(iterator);
    const float r = AiSqr(2 / width) * (AiSqr(offset.x) + AiSqr(offset.y));
    if (r > 1.0f)
        continue;

    // gaussian filter weight
    const float weight = AiFastExp(2 * -r) * inv_density;

    // accumulate weights and colors
    avalue += weight * AiAOVSampleIteratorGetRGBA(iterator);
    aweight += weight;
}

// compute final filtered color
if (aweight != 0.0f)
    avalue /= aweight;
*((AtRGBA*)data_out) = avalue;
}

node_loader
{
    if (i>0)
        return false;

    node->methods      = CustomGaussianFilterMtd;
    node->output_type  = AI_TYPE_NONE;
    node->name         = "custom_gaussian_filter";
    node->node_type    = AI_NODE_FILTER;
}

```

```
    strcpy(node->version, AI_VERSION);
    return true;
}
```

## Driver Nodes

- Simple Driver
- Display Driver
- Raw Driver

## Simple Driver

### Writing an output driver

An output driver gives you access to the one filtered value per pixel, after a bucket has been completely rendered. You can use drivers for example to write to new image file formats, or to send pixels to a display device, to an application's window, etc.

You can specify a driver in an .ass file like this:

```
outputs 1 1 STRING
"RGBA RGBA my_main_filter my_main_driver"
```

You can also specify multiple drivers for different AOVs, potentially using different filters and data types:

```
outputs 3 1 STRING
"RGBA RGBA my_filter my_main_driver"
"direct_diffuse RGBA my_filter my_direct_diffuse_driver"
"indirect_diffuse RGBA my_filter my_indirect_diffuse_driver"
```

The implementation of a sample driver that writes out to a file a list of all the objects in a pointer AOV would look like:

```
#include <ai.h>
#include <strings.h>
#include <fstream>
#include <unordered_map>

// This driver will write to a file a list of all the objects in a pointer AOV

AI_DRIVER_NODE_EXPORT_METHODS(DriverPtrMtd);
namespace ASTR {
    const AtString name("name");
    const AtString filename("filename");
}

typedef struct {
    std::unordered_map<AtString, AtNode*, AtStringHash> names;
} DriverPtrStruct;

node_parameters
{
    AiParameterStr(ASTR::filename, "objects.txt");
}

node_initialize
{
    DriverPtrStruct *driver = new DriverPtrStruct();
    // initialize the driver
    AiDriverInitialize(node, false);
    AiNodeSetLocalData(node, driver);
}

driver_needs_bucket
```

```

{
    return true;
}

driver_process_bucket
{ }

node_update
{ }

driver_supports_pixel_type
{
    // this driver will only support pointer formats
    return pixel_type == AI_TYPE_POINTER || pixel_type == AI_TYPE_NODE;
}

driver_open
{ // this driver is unusual and happens to do all the writing at the end, so this
function is
    // empty.
}

driver_extension
{
    static const char *extensions[] = { "txt", NULL };
    return extensions;
}

driver_prepare_bucket
{ }

driver_write_bucket
{
    DriverPtrStruct *driver = (DriverPtrStruct *)AiNodeGetLocalData(node);
    const void *bucket_data;
    // Iterate over all the AOVs hooked up to this driver
    while (AiOutputIteratorGetNext(iterator, NULL, NULL, &bucket_data))
    {
        for (int y = 0; y < bucket_size_y; y++)
        {
            for (int x = 0; x < bucket_size_x; x++)
            {
                // Get source bucket coordinates for pixel
                int sidx = y * bucket_size_x + x;
                // Because of driver_supports_pixel_type, we know pixel is a
                // pointer to an AtNode.
                AtNode* pixel_node = ((AtNode **)bucket_data)[sidx];
                const AtString name = AiNodeGetStr(pixel_node, ASTR::name);
                driver->names.emplace(name, pixel_node);
            }
        }
    }
}

driver_close
{
    DriverPtrStruct *driver = (DriverPtrStruct *)AiDriverGetLocalData(node);
}

```

```
    std::ofstream myfile(AiNodeGetStr(node, ASTR::filename));
    for (auto &i : driver->names)
        myfile << i.first << "\t " <<i.second << std::endl;
    myfile.close();
}

node_finish
{
    // Free local data
    DriverPtrStruct *driver = (DriverPtrStruct *)AiNodeGetLocalData(node);
    delete driver;
    AiDriverDestroy(node);
}

node_loader
{
    if (i>0)
        return false;
    node->methods = (AtNodeMethods*) DriverPtrMtd;
    node->output_type = AI_TYPE_NONE;
    node->name = "driver_ptr";
    node->node_type = AI_NODE_DRIVER;
```

```
    strcpy(node->version, AI_VERSION);
    return true;
}
```

## Display Driver

### Example

Simple display driver example, that passes 8 bit RGBA pixels to a specified callback.

#### driver\_display\_callback.cpp

```
#include <ai.h>

namespace ASTR {
    static const AtString callback("callback");
    static const AtString callback_data("callback_data");
    static const AtString color_space("color_space");
};

AI_DRIVER_NODE_EXPORT_METHODS(DriverDisplayCallbackMtd)

typedef void (*DisplayCallback)(uint32_t x, uint32_t y, uint32_t width, uint32_t
height, uint8_t* buffer, void* data);

node_parameters
{
    AiParameterPtr("callback"      , NULL   );
    AiParameterPtr("callback_data", NULL   ); // This value will be passed directly
to the callback function
}

node_initialize
{
    AiDriverInitialize(node, false);
}

node_update
{
}

driver_supports_pixel_type
{
    switch (pixel_type)
    {
        case AI_TYPE_FLOAT:
        case AI_TYPE_RGB:
        case AI_TYPE_RGBA:
            return true;
        default:
            return false;
    }
}

driver_extension
{
    return NULL;
}

driver_open
```

```

{
}

driver_needs_bucket
{
    return true;
}

driver_prepare_bucket
{
    DisplayCallback cb = (DisplayCallback) AiNodeGetPtr(node, ASTR::callback);

    // Call the callback function with a NULL buffer pointer, to indicate
    // a bucket is going to start being rendered.
    if (cb)
    {
        void *cb_data = AiNodeGetPtr(node, ASTR::callback_data);
        (*cb)(bucket_xo, bucket_yo, bucket_size_x, bucket_size_y, NULL, cb_data);
    }
}

driver_write_bucket
{
    int pixel_type;
    const void* bucket_data;

    // Get the first AOV layer
    if (!AiOutputIteratorGetNext(iterator, NULL, &pixel_type, &bucket_data))
        return;

    const bool dither = true;

    // Retrieve color manager for conversion
    AtNode* color_manager = (AtNode*)AiNodeGetPtr(AiUniverseGetOptions(),
"color_manager");
    AtString display_space, linear_space;
    AiColorManagerGetDefaults(color_manager, display_space, linear_space);
    if (!display_space)
        display_space = linear_space;

    // Allocates memory for the final pixels in the bucket
    //
    // This memory is not released here. The client code is
    // responsible for its release, which must be done using
    // the AiFree() function in the Arnold API
    uint8_t* buffer = (uint8_t*)AiMalloc(bucket_size_x * bucket_size_y *
sizeof(uint8_t) * 4);
    int minx = bucket_xo;
    int miny = bucket_yo;
    int maxx = bucket_xo + bucket_size_x - 1;
    int maxy = bucket_yo + bucket_size_y - 1;

    for (int j = miny; (j <= maxy); ++j)
    {
        for (int i = minx; (i <= maxx); ++i)
        {
            int bx = i - minx;

```

```

int by = j - miny;
AtRGBA source = AI_RGBA_ZERO;

switch (pixel_type)
{
    case AI_TYPE_FLOAT:
    {
        float f = ((float*)bucket_data)[by * bucket_size_x + bx];
        source = AtRGBA(f, f, f, 1.0f);
        break;
    }
    case AI_TYPE_RGB:
    {
        AtRGB rgb = ((AtRGB*)bucket_data)[by * bucket_size_x + bx];
        source = AtRGBA(rgb, 1.0f);
        break;
    }
    case AI_TYPE_RGBA:
    {
        source = ((AtRGBA*)bucket_data)[by * bucket_size_x + bx];
        break;
    }
}

AiColorManagerTransform(color_manager, display_space, false, false, NULL,
(uint8_t*)&source.rgb());

uint8_t* target = &buffer[(by * bucket_size_x + bx) * 4];
target[0] = AiQuantize8bit(i, j, 0, source.r, dither);
target[1] = AiQuantize8bit(i, j, 1, source.g, dither);
target[2] = AiQuantize8bit(i, j, 2, source.b, dither);
target[3] = AiQuantize8bit(i, j, 3, source.a, dither);
}

// Sends the buffer with the final pixels to the callback for display.
//
// The callback receives ownership over this buffer, so it must
// release it when it is done with it, using the AiFree() function
// in the Arnold API.
//
// The reason for doing this is to decouple this code from the visualization
// process, so, as soon as the buffer is ready, this driver will send it to
// the callback and return to the rendering process, which will continue
// asynchronously, in parallel with the visualization of the bucket, carried
// out by the client code.
//
DisplayCallback cb = (DisplayCallback) AiNodeGetPtr(node, ASTR::callback);
if (cb)
{
    void *cb_data = AiNodeGetPtr(node, ASTR::callback_data);
    (*cb)(bucket_xo, bucket_yo, bucket_size_x, bucket_size_y, buffer, cb_data);
}
}

driver_process_bucket
{

```

```
// Use this instead of driver_write_bucket for best performance, if your
// callback handling code is thread safe.
}

driver_close
{
}

node_finish
{
}

node_loader
{
    if (i>0)
        return false;

    node->methods      = DriverDisplayCallbackMtd;
    node->name         = "driver_display_callback";
    node->node_type    = AI_NODE_DRIVER;
```

```
    strcpy(node->version, AI_VERSION);
    return true;
}
```

## Raw Driver

In contrast to a regular output driver, which processes already filtered pixels, a "raw" driver gives you access to all the individual, unfiltered shading points along a ray, after a bucket has been completely rendered.

You can specify a raw driver in an .ass file like this:

```
outputs 2 1 STRING                                // this example has two drivers:  
    "RGBA RGBA my_main_filter my_main_driver"    //    1) regular driver for main RGBA  
output and its filter  
    "raw_driver_name"                            //    2) a raw driver
```

Note that you use a raw driver just by specifying its name; there is no filtering.

The implementation of a sample raw driver would look like this:

```
#include <ai.h>  
  
AI_DRIVER_NODE_EXPORT_METHODS(DriverRAWMtd);  
  
typedef struct {  
    ...  
} DriverRAWStruct;  
  
node_loader  
{  
    if (i>0)  
        return false;  
    node->methods = (AtNodeMethods*) DriverRAWMtd;  
    node->output_type = AI_TYPE_NONE;  
    node->name = "driver_raw";  
    node->node_type = AI_NODE_DRIVER;  
    strcpy(node->version, AI_VERSION);  
    return true;  
}  
  
node_parameters  
{  
    AiParameterStr( "filename", "deep.raw" );  
}  
  
node_initialize  
{  
    DriverRAWStruct *raw = new DriverRAWStruct();  
  
    static const char *required_aovs[] = { "FLOAT Z", "FLOAT A", NULL };  
  
    // Initialize the driver (set the required AOVs and indicate that  
    // we want values at all the depths)  
    AiRawDriverInitialize(node, required_aovs, true /* deep driver */);  
    AiNodeSetLocalData(raw);  
}  
  
node_update
```

```

{
    ...
}

driver_supports_pixel_type
{
    // This function is not needed for raw drivers
    return true;
}

driver_open
{
    ...
}

driver_extension
{
    static const char *extensions[] = { "raw", NULL };
    return extensions;
}

driver_needs_bucket
{
    return true;
}

driver_prepare_bucket
{
    ...
}

driver_process_bucket
{
    ...
}

driver_write_bucket
{
    DriverRAWStruct *raw = (DriverRAWStruct *)AiDriverGetLocalData(node);
    ...

    for (y = bucket_yo; y < bucket_yo + bucket_size_y; y++)
    {
        for (x = bucket_xo; x < bucket_xo + bucket_size_x; x++)
        {
            // Iterator for samples in this pixel
            AiAOVSampleIteratorInitPixel(sample_iterator, x, y);
            while (AiAOVSampleIteratorGetNext(sample_iterator))
            {
                AtVector2 position = AiAOVSampleIteratorGetOffset(sample_iterator);
                while (AiAOVSampleIteratorGetNextDepth(sample_iterator))
                {
                    float a = AiAOVSampleIteratorGetAOVFlt(sample_iterator, "A");
                    float z = AiAOVSampleIteratorGetAOVFlt(sample_iterator, "Z");
                    ...
                }
            }
        }
    }
}

```

```
        }
    }
    ...
}

driver_close
{
    ...
}

node_finish
```



## Rendering

- How to Read a Render Log
- Farm Rendering
- Removing Noise
- Removing Noise Workflow
- What is Sampling?

## How to Read a Render Log

The Render Log prints the progress messages, warnings and errors to the console or a file as kick renders an image. As well as showing the percentage completed, these detailed statistics can be used to optimize and debug scenes.

The Arnold log provides detailed statistics that are useful for debugging, optimizing and benchmarking renders. It is the first thing to examine should you encounter errors and usually the first information to send to support.

When encountering problems with renders, the first thing you should do with a log file is search for the word "**ERROR**". Most of the time you will find the source of the problem in that ERROR message.

The verbosity levels vary from kick to the plugins.

Kick	Plugins
0 - No Output	Errors
1 - Progress	
2	Warnings + Info
3	
4	Debug
5 - Detailed	
6 - Debug	

The log below contains the most common output. The first column of the log, `00:00:00`, shows the time elapsed in `hh:mm:ss`. The second column, `773MB`, shows the memory usage at that stage.

### **Initialization (Info)**

This section shows the version of Arnold being used and its build details. It also lists the hardware specifications and operating system of the machine it was rendered on.

```
00:00:00 773MB | log started Tue Jul 21 15:26:25 2015
00:00:00 773MB | Arnold 4.2.7.4 windows icc-14.0.2 oiio-1.5.15 rlm-11.2.2
2015/06/15 09:39:31
00:00:00 773MB | host application: MtoA 1.2.3.1 03a85380bec8 (Master)
MtoA-1.2.3.1 Maya 2016
00:00:00 773MB | running on PC, pid=12188
00:00:00 773MB | 1 x Intel(R) Xeon(R) CPU E5-1650 v2 @ 3.50GHz (6 cores,
12 logical) with 32712MB
00:00:00 773MB | Windows 7 Professional Service Pack 1 (version 6.1,
build 7601)
```

There is a single initialization process for the whole render session (only on the first render), with an updated process for every following render.

```
00:00:00 773MB | [mtoa.session]    Initializing at frame 1.000000
00:00:00 773MB | [mtoa] Exporting Arnold options
'defaultArnoldRenderOptions'
00:00:00 773MB | [mtoa.extensions] aiOptions Using translator
<built-in>, provided by <built-in>(<built-in>).
00:00:00 773MB | [mtoa.session]    defaultArnoldRenderOptions
Exporting plug defaultArnoldRenderOptions.message for type aiOptions
```

## **Loading / Plugins**

This section displays the results of importing any shaders or procedurals.

Just rendering through kick on its own shows no plugin and the ass location.

```
00:00:00 773MB | loading plugins from . . .
00:00:00 773MB | no plugins loaded
00:00:00 773MB | [ass] loading project/scenes/simplescene.ass ...
00:00:00 773MB | [ass] read 11386 bytes, 11 nodes in 0:00.00
```

If a plugin is being loaded (and verbosity is high enough) the shaders will be listed:

```
00:00:00 773MB | loading plugin:
C:/solidangle/mtoadeploy/2016/shaders/mtoa_shaders.dll ...
00:00:00 773MB | mtoa_shaders.dll: MayaMultiplyDivide uses Arnold
4.2.7.4
00:00:00 773MB | mtoa_shaders.dll: MayaClamp uses Arnold 4.2.7.4
00:00:00 773MB | mtoa_shaders.dll: MayaGammaCorrect uses Arnold 4.2.7.4
00:00:00 773MB | mtoa_shaders.dll: MayaCondition uses Arnold 4.2.7.4
...
00:00:00 774MB | mtoa_shaders.dll: volume_sample_float uses Arnold
4.2.7.4
00:00:00 774MB | mtoa_shaders.dll: volume_sample_rgb uses Arnold 4.2.7.4
00:00:00 774MB | mtoa_shaders.dll: driver_mplay uses Arnold 4.2.7.4
00:00:00 774MB | loaded 90 plugins from 1 lib(s) in 0:00.00
```

## **License Check**

If a license is available, the log will display the following.

```
00:00:00 810MB | [rlm] checkout of "arnold 20150615" in progress ...
00:00:00 810MB | [rlm] checkout of "arnold 20150615" from server PC in
0:00.01
00:00:00 810MB | [rlm] expiration date: 31-dec-2015 (164 days left)
```

However, if the RLM server is not running, it will display a warning.

```
00:00:00 810MB WARNING | [rlm] could not connect to license server on
5053@localhost
```

## **Color Management**

```

00:00:00    56MB      |
00:00:00    63MB      | [color_manager_syncolor] Using syncolor_color_manager
Version 2018.0.80
00:00:00    63MB      |
directory from /path/to/synColor
00:00:00    63MB      |
directory from /other/path/to/Shared/

```

When using the SYNCOLOR environment variable, kick will override the ass color management paths so the output would be:

```

00:00:00    63MB      | [color_manager_syncolor] Using syncolor_color_manager
Version 2018.0.80
00:00:00    63MB      |
/path/to/synColorConfig.xml
00:00:00    63MB      |
directory from /other/path/to/Shared/

```

## Scene Contents

Here it will list the numbers of lights and objects (and their types) in the scene.

```

00:00:00    818MB      | there are 1 light and 2 objects:
00:00:00    818MB      |     1 persp_camera
00:00:00    818MB      |     1 distant_light
00:00:00    818MB      |     1 utility
00:00:00    818MB      |     1 lambert
00:00:00    818MB      |     1 driver_exr
00:00:00    818MB      |     1 gaussian_filter
00:00:00    818MB      |     1 polymesh
00:00:00    818MB      |     1 list_aggregate
00:00:00    818MB      |     1 MayaShadingEngine
00:00:00    818MB      |     1 renderview_display
00:00:00    818MB      |     1 color_manager_syncolor

```

## Resolution / Samples

This section displays the output resolution and details about the number of samples.

```

00:00:00    818MB      | rendering image at 640 x 480, 3 AA samples
00:00:00    818MB      | AA sample clamp <disabled>
00:00:00    818MB      | diffuse           samples 3 / depth 1
00:00:00    818MB      | specular          samples 3 / depth 2
00:00:00    818MB      | transmission       samples 1 / depth 2
00:00:00    818MB      | volume indirect   <disabled by depth>
00:00:00    818MB      | total              depth 10
00:00:00    818MB      | bssrdf             <disabled>
00:00:00    818MB      | transparency        depth 10 / fast opacity off

```

## Nodes

Before rendering starts, nodes are initialized. The stats here give insight into the objects and lights in the scene and can show warnings if anything is configured incorrectly. Note that some data is not immediately initialized, some computations are delayed until the first ray hits the object so that further information can be shown during the render.

```
00:00:00 818MB | initializing 11 nodes ...
00:00:00 818MB | creating root object list ...
00:00:00 818MB | scene bounds: (-0.995465517 -1.01209259 -0.995465755)
-> (1.00453472 0.98790741 1.00453484)
00:00:00 818MB | node initialization done in 0:00.00 (single-threaded)
00:00:00 818MB | updating 12 nodes ...
00:00:00 818MB | directionalLightShape1: distant_light using 1 sample
00:00:00 818MB | node update done in 0:00.00 (single-threaded)
```

## Drivers / AOVs

Here are the drivers and AOVs used in the scene, and any warnings if they are set up incorrectly.

```
00:00:00 818MB | [aov] parsing output statement: "RGBA RGBA
defaultArnoldFilter@gaussian_filter defaultArnoldDisplayDriver@renderview_display"
00:00:00 818MB | [aov] parsing output statement: "RGBA RGBA
defaultArnoldFilter@gaussian_filter defaultArnoldDriver@driver_exr.RGBA"
00:00:00 818MB | [aov] registered driver:
"defaultArnoldDisplayDriver@renderview_display" (renderview_display)
00:00:00 818MB | [aov] * "RGBA" of type RGBA filtered by
"defaultArnoldFilter@gaussian_filter" (gaussian_filter)
00:00:00 818MB | [aov] registered driver:
"defaultArnoldDriver@driver_exr.RGBA" (driver_exr)
00:00:00 818MB | [aov] * "RGBA" of type RGBA filtered by
"defaultArnoldFilter@gaussian_filter" (gaussian_filter)
00:00:00 818MB | [aov] done preparing 1 AOV for 2 outputs to 2 drivers
(0 deep AOVs)
```

## Progress

Now that the render is starting, the log will show how many buckets are being used and at what size. If any importance tables need to be generated, in this case for the skydome, they will be done so now, showing the average energy value. Percentage completed is shown in 5% increments along with the average rays per pixel. This value will show at a glance if your sampling settings are too high.

```

00:00:00 825MB | starting 12 bucket workers of size 64x64 ...
00:00:00 835MB | 0% done - 9 rays/pixel
00:00:00 835MB | [accel] bvh4 done - 0:00.00 - 400 prims, 1 key
00:00:00 836MB | 5% done - 9 rays/pixel
00:00:00 836MB | 10% done - 9 rays/pixel
00:00:00 836MB | 15% done - 9 rays/pixel
00:00:00 837MB | 20% done - 9 rays/pixel
00:00:00 837MB | 25% done - 9 rays/pixel
00:00:00 837MB | 30% done - 9 rays/pixel
00:00:00 837MB | 35% done - 10 rays/pixel
00:00:00 837MB | 40% done - 10 rays/pixel
00:00:00 838MB | 45% done - 12 rays/pixel
00:00:00 838MB | 50% done - 12 rays/pixel
00:00:00 838MB | 55% done - 13 rays/pixel
00:00:00 838MB | 60% done - 13 rays/pixel
00:00:00 838MB | 65% done - 13 rays/pixel
00:00:00 838MB | 70% done - 13 rays/pixel
00:00:00 838MB | 75% done - 13 rays/pixel
00:00:00 838MB | 80% done - 13 rays/pixel
00:00:00 838MB | 85% done - 13 rays/pixel
00:00:00 838MB | 90% done - 13 rays/pixel
00:00:00 838MB | 95% done - 13 rays/pixel
00:00:00 838MB | 100% done - 13 rays/pixel
00:00:00 824MB | bucket workers done in 0:00.38

```

## Output

Once the render is complete, it will be written to file (or screen) using the selected output driver. The log will show the relative path.

```

00:00:00 824MB | [driver_exr] writing file
`C:/Users/Documents/example.exr'
00:00:00 821MB | render done

```

## Scene Creation Stats

These timings show the time taken to load plugins and .ass files.

```

00:00:00 821MB | scene creation time:
00:00:00 821MB | plugin loading           0:00.07
00:00:00 821MB | system/unaccounted        0:00.12
00:00:00 821MB | total                      0:00.12 ( 1.05% machine
utilization)

```

## Render Time Stats

These timings show the time spent performing various tasks. Typically pixel rendering will take up most of the time if that's not the case then looking at the node initialization and geometry stats might reveal inefficiencies. Low machine utilization may indicate that other processes on the same machine are slowing down the render. But it can also indicate that a lot of time is spent performing single threaded tasks, for example in node initialization, procedurals or mesh processing. Another possibility is that there is a lot of (slow) file access, typically for textures or volumes. Looking at the texture stats might reveal problems.

00:00:00	821MB	render time:	
00:00:00	821MB	node init	0:00.00
00:00:00	821MB	sanity checks	0:00.00
00:00:00	821MB	bucket rendering	0:00.38
00:00:00	821MB	mesh processing	0:00.00
00:00:00	821MB	accel. building	0:00.00
00:00:00	821MB	pixel rendering	0:00.38 (multi-threaded render, this value may not be reliable)
00:00:00	821MB	system/unaccounted	0:00.17
00:00:00	821MB	total	0:00.55 (69.23% machine utilization)

### **Memory Stats**

Here memory used at startup and peak memory for various types of data are listed. When using a plugin, the startup memory shows how much memory the host application is using, when rendering from kick the startup memory is small which typically means larger scenes can be rendered. If memory usage is too high, these stats indicate which data would be most helpful to reduce.

00:00:00	821MB	memory consumed in MB:	
00:00:00	821MB	at startup	807.25
00:00:00	821MB	plugins	2.52
00:00:00	821MB	AOV samples	11.67
00:00:00	821MB	output buffers	5.32
00:00:00	821MB	node overhead	0.00
00:00:00	821MB	message passing	0.02
00:00:00	821MB	memory pools	13.54
00:00:00	821MB	geometry	0.01
00:00:00	821MB	polymesh	0.01
00:00:00	821MB	accel. structs	0.02
00:00:00	821MB	strings	0.30
00:00:00	821MB	texture cache	0.00
00:00:00	821MB	unaccounted	1.64
00:00:00	821MB	total peak	842.29

### **Ray Stats**

These are the number of rays of each type, per pixel and per AA sample. If the render time is high, these numbers give insight into which types of samples are most expensive to render and would be most helpful trying to reduce.

00:00:00	821MB	ray counts:	(/pixel , /sample)
(% total)	(avg. hits)	(max hits)	
00:00:00	821MB	camera	2318256 ( 8.71, 1.00)
( 64.57%)	( 0.11)	( 1)	
00:00:00	821MB	shadow	228076 ( 0.86, 0.10)
( 6.35%)	( 0.00)	( 0)	
00:00:00	821MB	diffuse	1043914 ( 3.92, 0.45)
( 29.08%)	( 0.00)	( 0)	
00:00:00	821MB	total	3590246 ( 13.48, 1.55)
(100.00%)	( 0.07)	( 1)	
00:00:00	821MB	max depth	1

### **Shader Stats**

The number of shader calls in various contexts is shown here, which can be useful to figure out which shader calls are most helpful to speed up.

- Primary: surface shading of non-shadow rays (this is traditional shading).
- Transparent\_shadow: surface shading of shadow rays when it goes through a transparent surface (needed to shade to know if the ray is blocked or can continue, and if it continues, has its color changed).
- Autobump: each time autobump is computed, it has to compute displacement.
- Background: the background shader.
- Importance: calls made while computing importance tables.
- Volume: volume shader calls.

00:00:00	821MB	shader calls:	(/pixel , /sample)
		(% total)	
00:00:00	821MB	primary	523006 ( 1.96 , 0.23 )
(100.00%)			
00:00:00	821MB	total	523006 ( 1.96 , 0.23 )
(100.00%)			

You can use these shader stats to improve render times. If for instance, you see lots of transparent\_shadow, that is usually an indication that your light samples are very high or you have lots of transparent objects. Another example would be if autobump is very high, you could try disabling autobump on meshes that don't seem to benefit from it as much.

## Geometry Stats

These stats give insight into the amount of geometry in the scene. If memory usage is high or scene setup takes a long time, these can be used to find the objects that contribute to it most, and would benefit from being simplified or having their subdivision iterations reduced.

```

00:00:00 821MB | geometry: (% hit )
(instances) ( init mem, final mem)
00:00:00 821MB | lists 1 (100.0%)
0) ( 0.00, 0.00)
00:00:00 821MB | polymeshes 1 (100.0%)
0) ( 0.02, 0.01)
00:00:00 821MB |
-----
-----+
00:00:00 821MB | geometric elements: ( min) (
avg.) ( max)
00:00:00 821MB | objects (top level) 1 ( 1) (
1.0) ( 1)
00:00:00 821MB | polygons 400 ( 400) (
400.0) ( 400)
00:00:00 821MB |
-----
-----+
00:00:00 821MB | triangle tessellation: ( min) (
avg.) ( max) (/ element) (% total)
00:00:00 821MB | polymeshes 760 ( 760) (
760.0) ( 760) ( 1.90) (100.00%)
00:00:00 821MB | unique triangles: 760
00:00:00 821MB | memory use (in MB) 0.01
00:00:00 821MB | vertices 0.00
00:00:00 821MB | vertex indices 0.00
00:00:00 821MB | packed normals 0.00
00:00:00 821MB | normal indices 0.00
00:00:00 821MB | uv coords 0.00
00:00:00 821MB | uv coords idxs 0.00
00:00:00 821MB | uniform indices 0.00
00:00:00 821MB | userdata 0.00
00:00:00 821MB | largest polymeshes by triangle count:
00:00:00 821MB | 760 tris -- pSphereShape1

```

### Texture Stats

Detailed statistics for image textures. These give insight into which textures use most memory, and which textures are untiled and would benefit from being converted to .tx files. The percentage of the main cache misses should be very small for good performance, if it is high render time can be significantly increased. If the main cache misses are too high, it helps to ensure only .tx files are used, reduce the number of textures, or tweak the texture settings.

```

00:00:00 841MB | OpenImageIO Texture statistics
00:00:00 841MB |   Queries/batches :
00:00:00 841MB |     texture      : 261503 queries in 261503 batches
00:00:00 841MB |     texture 3d   : 0 queries in 0 batches
00:00:00 841MB |     shadow       : 0 queries in 0 batches
00:00:00 841MB |     environment  : 0 queries in 0 batches
00:00:00 841MB |   Interpolations :
00:00:00 841MB |     closest      : 0
00:00:00 841MB |     bilinear    : 398256
00:00:00 841MB |     bicubic     : 701789
00:00:00 841MB |   Average anisotropic probes : 2.8
00:00:00 841MB |   Max anisotropy in the wild : 671
00:00:00 841MB |
00:00:00 841MB | OpenImageIO ImageCache statistics (000000003D574040) ver
1.5.15
00:00:00 841MB |   Images : 1 unique
00:00:00 841MB |   ImageInputs : 1 created, 1 current, 1 peak
00:00:00 841MB |   Total size of all images referenced : 1.7 MB
00:00:00 841MB |   Read from disk : 1.7 MB
00:00:00 841MB |   File I/O time : 0.2s (0.0s average per thread)
00:00:00 841MB |   File open time only : 0.0s
00:00:00 841MB |   Tiles: 25 created, 24 current, 24 peak
00:00:00 841MB |   total tile requests : 1454328
00:00:00 841MB |   micro-cache misses : 35383 (2.43294%)
00:00:00 841MB |   main cache misses : 25 (0.00171901%)
00:00:00 841MB |   Peak cache memory : 2.3 MB
00:00:00 841MB |   Image file statistics:
00:00:00 841MB |       opens   tiles   MB read   I/O time   res
File
00:00:00 841MB |       1       1      12      1.7      0.2s    768x 768x3.u8
C:/Users/Documents/example.tif UNTILED UNMIPPED MIP-COUNT [12,7,3,2,1,0,0,0,0,0]
00:00:00 841MB |
00:00:00 841MB |   Tot:      1       12      1.7      0.2s
00:00:00 841MB |   1 not tiled, 1 not MIP-mapped

```

## Shutdown

When Arnold is finished, it will release any resources, memory/threads, and shutdown.

```

00:00:00 841MB | releasing resources
00:00:00 831MB | unloading 27 plugins
00:00:00 829MB | unloading plugins done
00:00:00 828MB | Arnold shutdown

```

## Farm Rendering

Some things to take into account when rendering in a farm or queue:

- -dp should always be present in your command line; otherwise progressive renders will waste time. This triggers an error in any case.
- Memory overcommitting should be either fully enabled in Linux (`echo 1 > /proc/sys/vm/overcommit_memory`), or set to the default mode of partially enabled according to a heuristic (`echo 0 > /proc/sys/vm/overcommit_memory`). If disabled (always the case in Windows and configurable in Linux), then the swap file should be made at least 2-3x larger than physical RAM on the machine so that Arnold can use all the RAM. Otherwise, without overcommitted memory enabled, Arnold, will run out of memory and crash before it can use all of the physical memory on the machine. For instance, it is theoretically possible for Arnold to run out of memory using just 12GB of RAM when the machine has 24GB of RAM if overcommit is disabled and the swap file is set to 0GB. Note that the swap file will not be used, and performance will not suffer until all of the RAM on the machine is used.

# Removing Noise

## Types Of Noise

The first step in removing noise from your renders is to identify where it is coming from. Noise can be caused by:

### **Insufficient Sampling:**

- **Motion Blur**
- **Depth Of Field**
- **Diffuse**
- **Specular**
- **Shadow**
- **Indirect Specular**
- **Transmission**
- **SSS**
- **Atmospheric Volume**

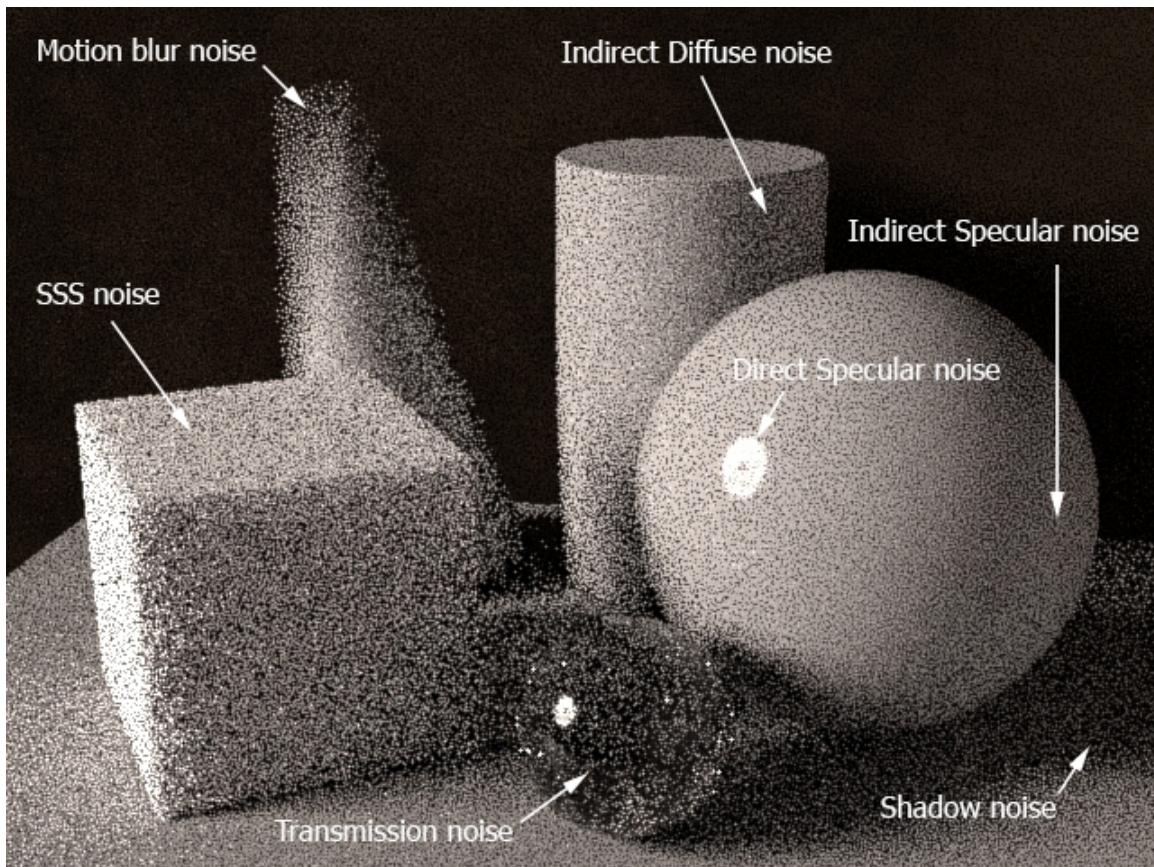
### **Other:**

- **Fireflies**
- **Non-energy conserving shaders, networks or settings**

Noise nearly always comes from insufficient sampling, but increasing sampling for the wrong rays can make the render times increase without helping to remove the noise. As an artist is usually working to a render time limit or amount of rays the aim is to allocate those rays as effectively as possible to minimize the noise in the most efficient manner. So if the *Camera (AA)* samples have to be increased to remove DOF noise, the other settings must be lowered to keep render times manageable. However, if DOF or motion blur is not a concern, then increasing *Camera (AA)* samples would fix all noise elsewhere but would also slow render times from the unnecessary rays.

For a procedure for removing noise, follow the **Removing Noise Workflow** diagram.

Here is a very noisy scene where all of the samples have been set to 1.

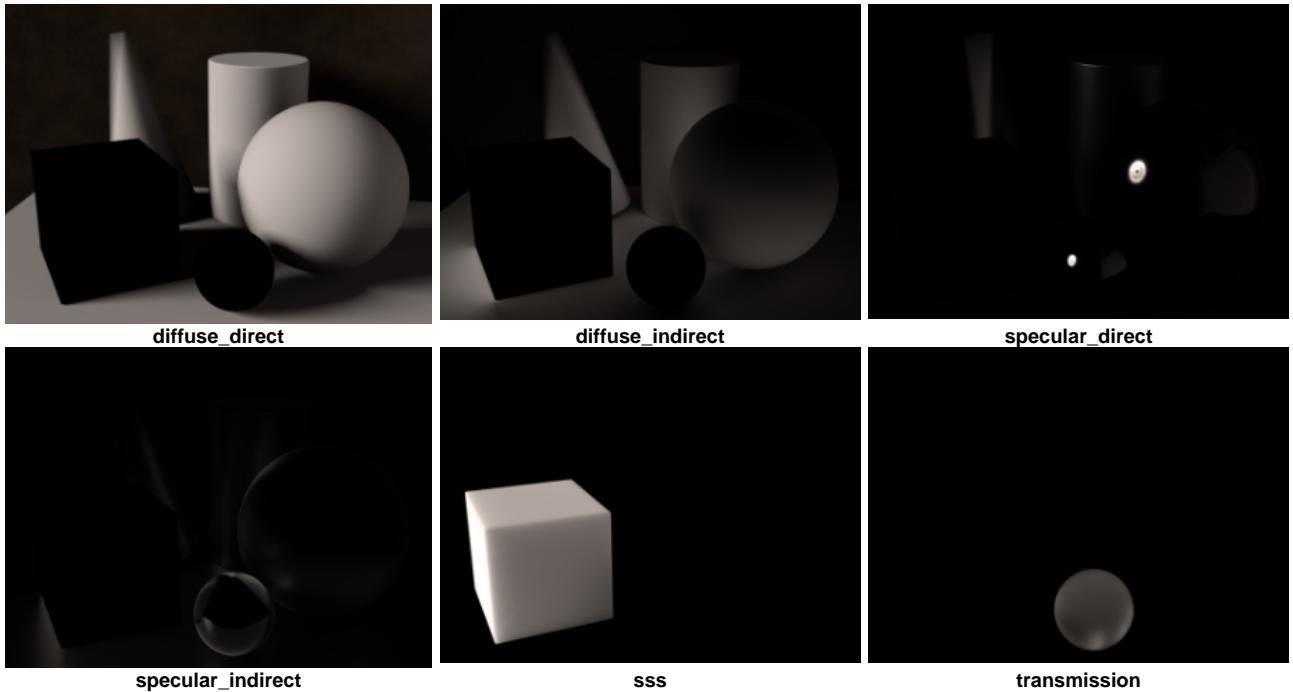


All samples: 1 (rollover image for final sample settings)

- **Camera (AA) samples:** 1
- **Diffuse samples:** 1
- **Specular samples:** 1
- **Transmission samples:** 1
- **SSS samples:** 1
- **Light samples:** 1

The most efficient method for identifying noise is to render AOVs. This can help save time by reducing the need to adjust settings unnecessarily.

## AOVs

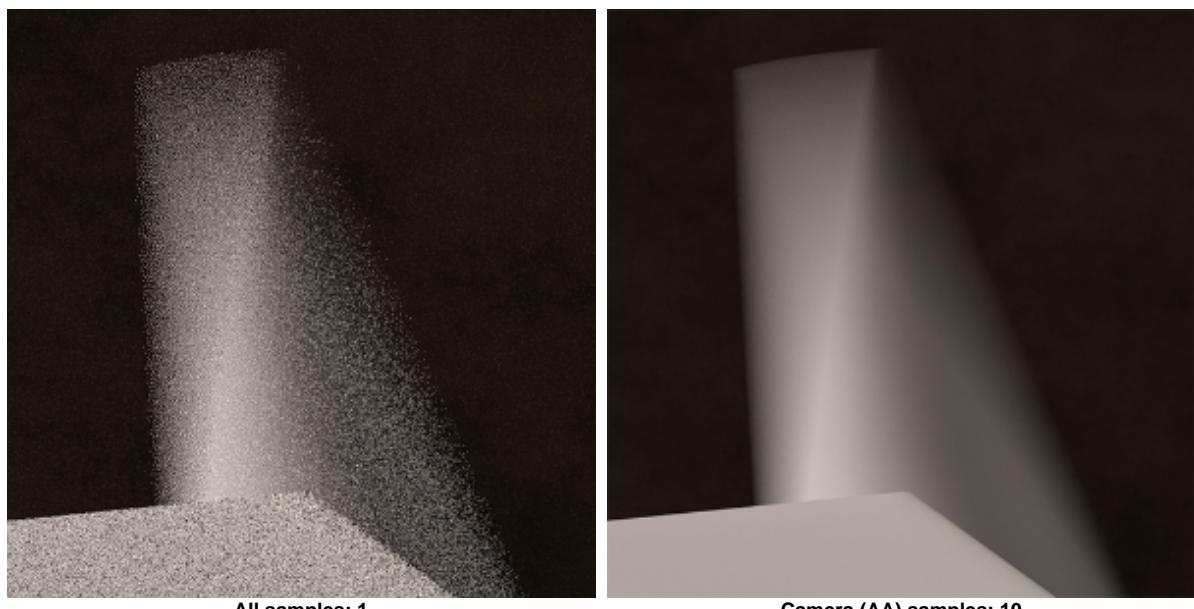


In the examples below the smoother option is shown with samples of 10. This is a very high value and is only used here because the Camera (AA) samples and all other samples are at 1 for an illustrative effect to contrast the noisy and smooth areas. Normally the Camera (AA) samples would be around 4-8, and as it acts as a multiplier, similar values would be needed for the other samples.

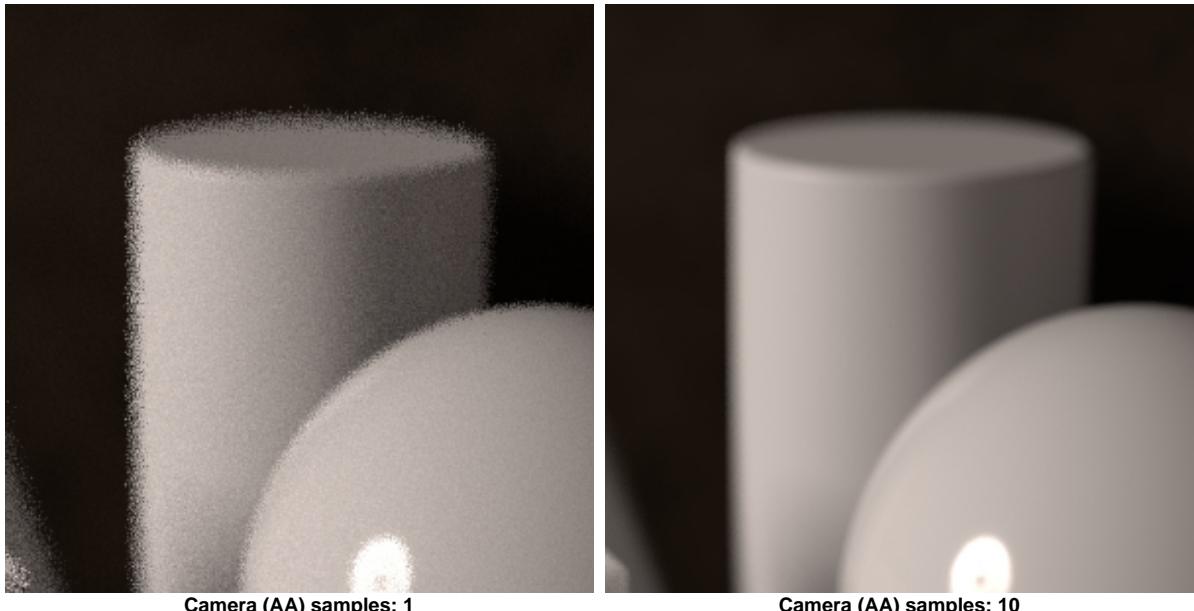
### ***Motion Blur/Depth Of Field Noise***

Motion blur noise shows up in the trails of moving geometry while DOF noise appears in the out of focus areas.

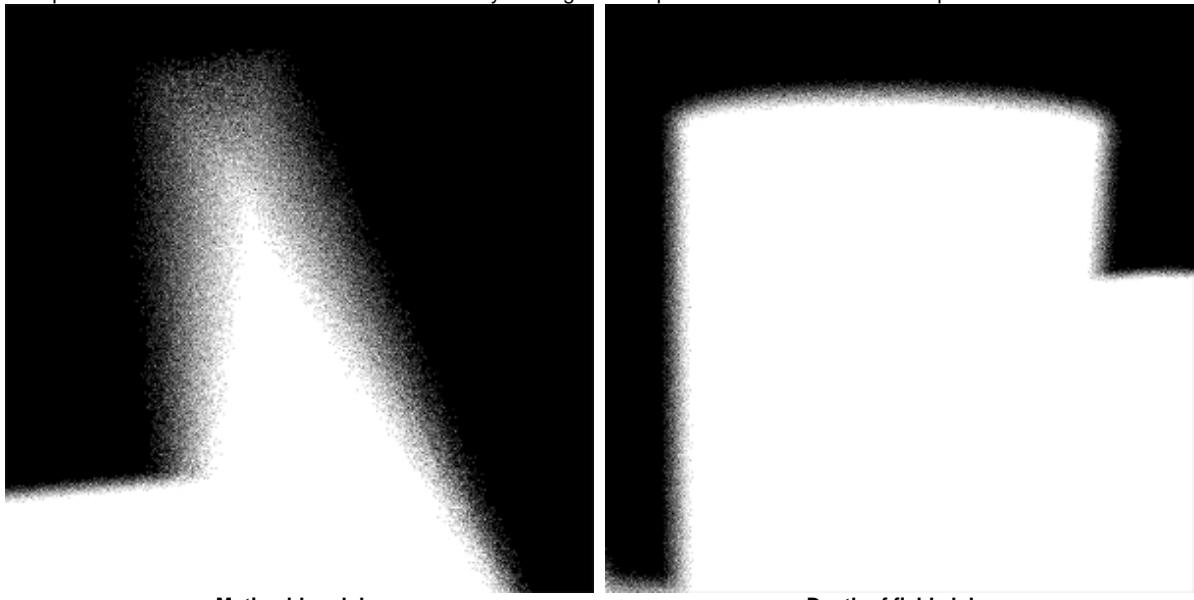
#### ***Motion Blur***



#### ***Depth Of Field***



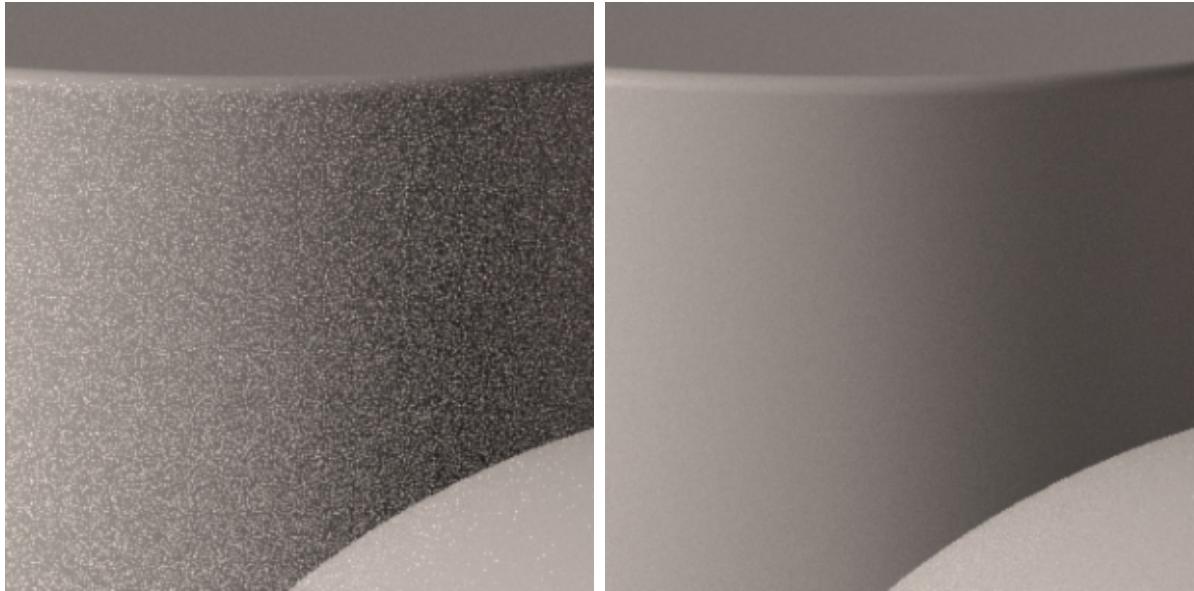
Both depth of field and motion blur can be confirmed by looking at the alpha channel to see if noise is present.



Motion blur and depth of field noise are caused by insufficient camera rays and therefore can only be solved by increasing *Camera (AA)* samples. The actual number of *Camera (AA)* samples is the square of this number. *Camera (AA)* samples of 4 results in 16 rays being cast. Note that increasing *Camera (AA)* samples will also increase the other samples meaning that they should be decreased to compensate.

Increasing the *Camera (AA)* samples will have a dramatic effect on render times. If motion blur or DOF is not a problem, then *Camera (AA)* samples should be the last consideration for fixing the other noise types.

#### ***Indirect Diffuse Noise***



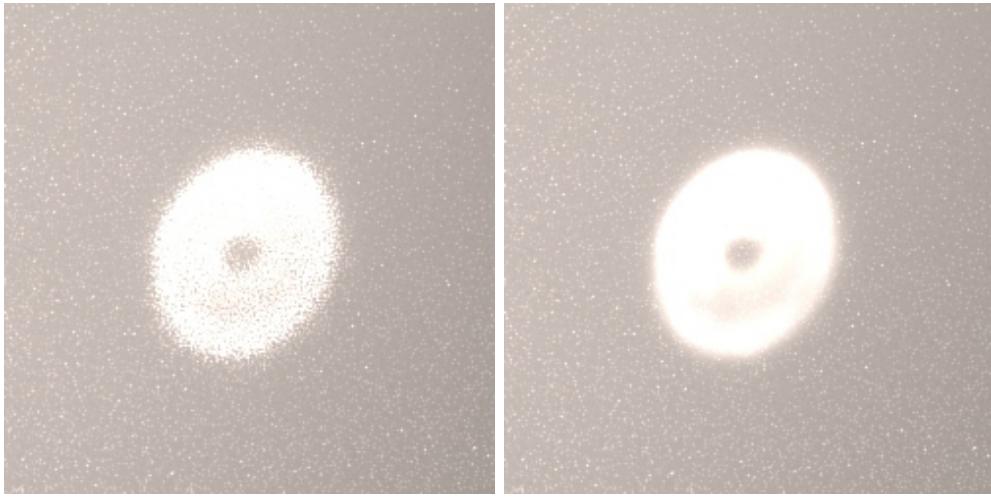
All samples: 1

Diffuse samples: 10.

Indirect diffuse noise is a common type of noise. The easiest way to confirm the cause is to check the indirect diffuse AOV. Another method is to set *Diffuse* samples to 0 which will turn off indirect diffuse. If the noise disappears, then it is created by indirect diffuse.

When *Diffuse* samples are more than zero, camera rays intersecting with diffuse surfaces fire indirect diffuse rays. The rays are fired in random directions within a hemispherical spread. Noise is introduced when there are insufficient rays to resolve the range of values from the environment. The noise can be removed by increasing *Diffuse* samples.

### Indirect Specular Noise



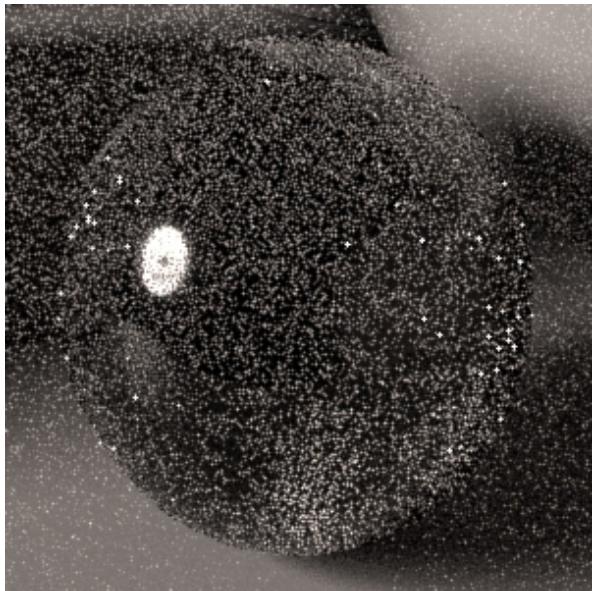
All samples: 1

Specular samples: 10

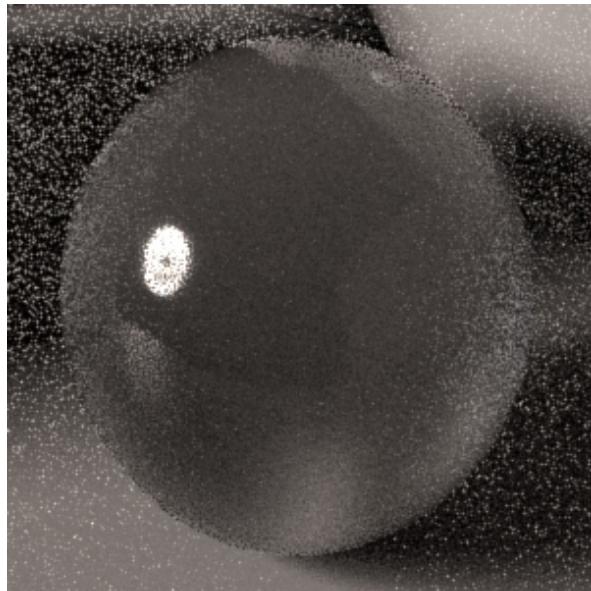
Indirect specular noise occurs when the *Specular Roughness* parameter > 0. The easiest way to confirm the cause is to check the indirect specular AOV. Another method is to set *Specular* samples to 0 which will remove blurred reflections. If the noise disappears, then it is created by indirect specular.

Indirect specular noise is caused by a lack of *Specular* samples. These samples control the number of rays fired when computing the reflected indirect-radiance integrated over the hemisphere weighted by a specular BRDF. The exact number of rays is the square of this value. Increase this number to reduce the indirect specular noise. Remember that the sampling is done for each *Camera (AA)* sample, so high values for both *Camera (AA)* samples and *Specular samples* will tend to result in slow renders.

### Transmission Noise



All samples: 1

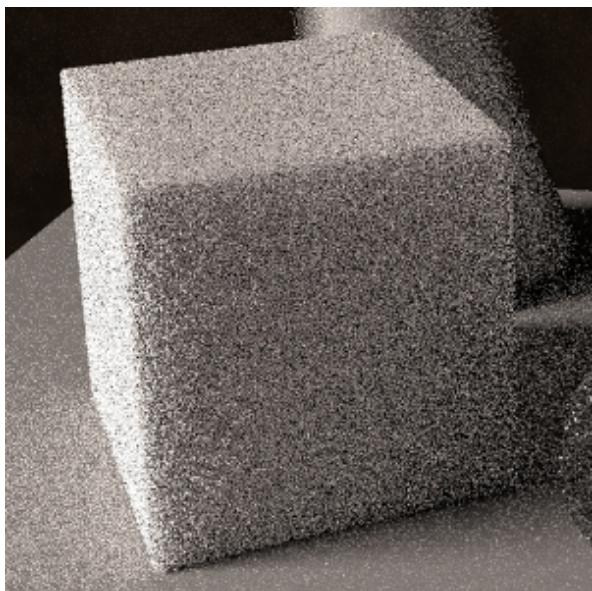


Transmission samples: 10. All other samples: 1.

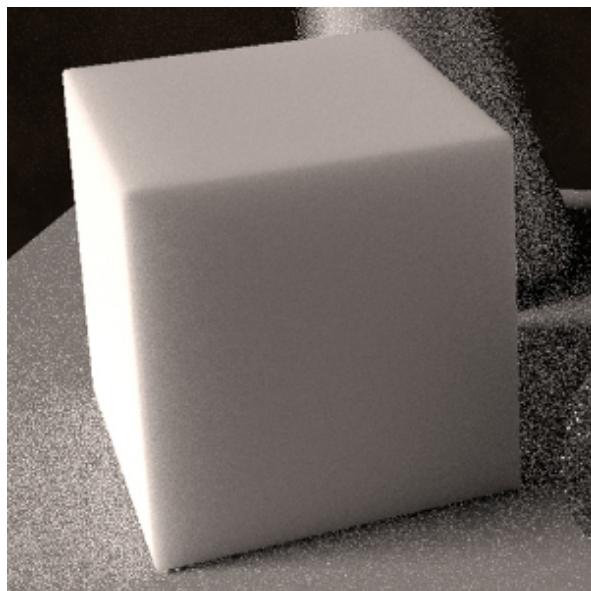
*Transmission* noise is noticeable in the blurred refraction of a transparent object when *Specular Roughness* > 0. The easiest way to confirm this is to check the *Transmission* AOV. Another method is to set *Transmission Samples* to 0 which will remove blurred refractions. If the noise disappears, then it is created by *Transmission*.

*Transmission* noise is caused by a lack of *Transmission* samples. *Transmission* samples control the number of samples used to simulate the microfacet-based glossy refraction evaluations. The exact number of rays is the square of this value. Increase this number to reduce noise in shaders using *Transmission*.

### SSS Noise



All samples: 1

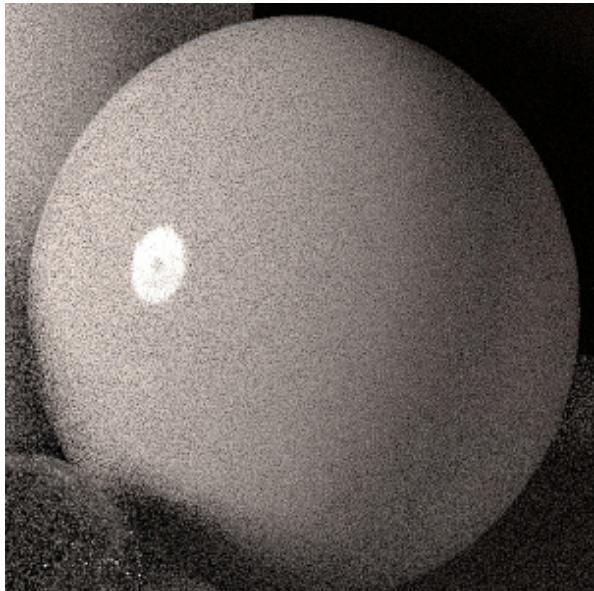


SSS samples: 10

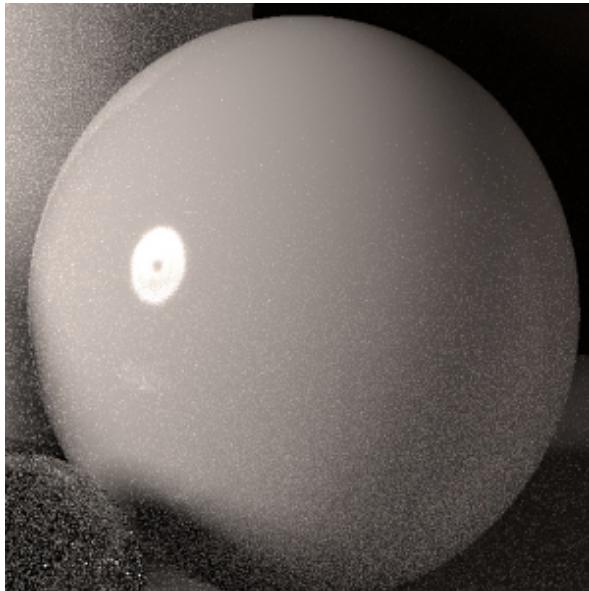
Subsurface-scattering (SSS) noise will occur on surfaces that are using the *Standard Surface* shader with SSS enabled. To confirm, check the SSS AOV. The SSS noise can be removed by increasing SSS samples.

### Direct Specular/Shadow Noise

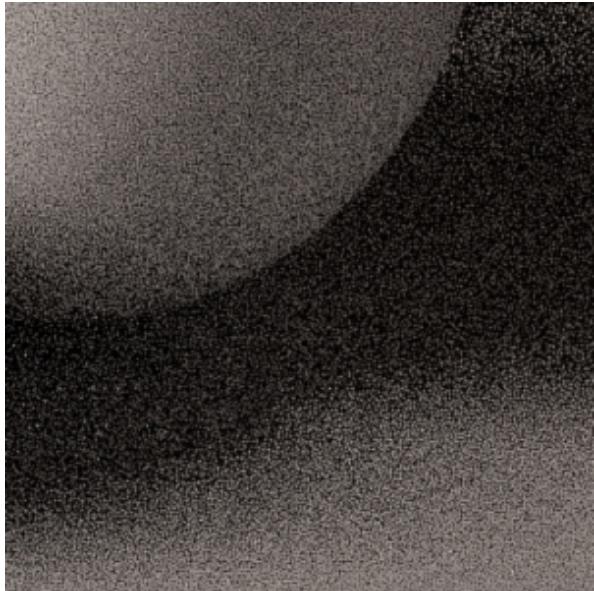
Noise in the direct specular and shadow is caused by a lack of light samples. Normally a small number of samples will be required to remove direct specular noise, but more samples may be required to remove shadow noise. The larger the radius a light source has, the softer the shadows will be, and therefore more samples will be required to remove shadow noise.



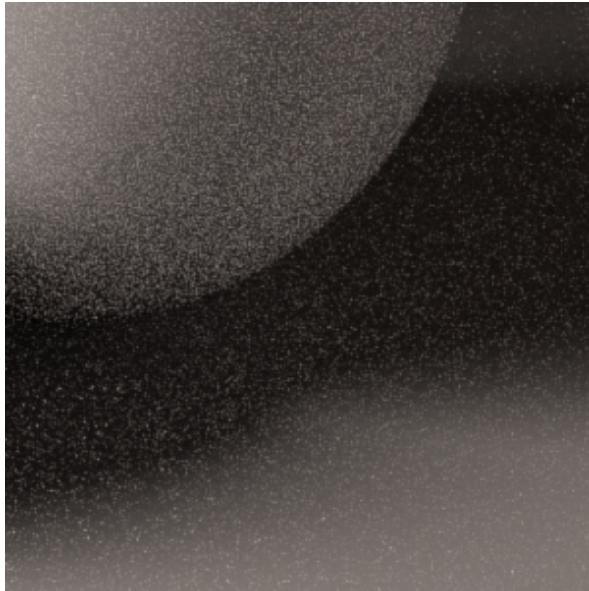
Light samples: 1



Light samples: 10



Light samples: 1



Light samples: 10

Light samples between 3-4 should be sufficient for most scenes.

Shadow noise is often mixed up with indirect diffuse noise, particularly for lights with large radii because their shadows will be softer. Look at the *Direct Diffuse AOV* to see if the shadows are noisy. Alternatively set *Diffuse* samples and *Specular* samples to 0. This will remove GI which will allow you to single out the direct lighting contribution. It can be tricky to identify direct specular noise as it could be mistaken for indirect specular noise. Direct specular is the reflection of the light itself on the surface. This will help you distinguish it from the reflections of surrounding objects based on color and intensity.

Noise can also appear on the specular highlights of thin geometry that use a shader with a high *Specular Weight* value. Increasing the number of *Camera (AA)* samples can help reduce this effect to a point. However, even with high *Camera (AA)* samples, anti-aliasing artifacts can appear. Increasing the *Specular Roughness* value helps to reduce this type of noise. Increasing the size of the light and reducing its intensity can also help in these situations.

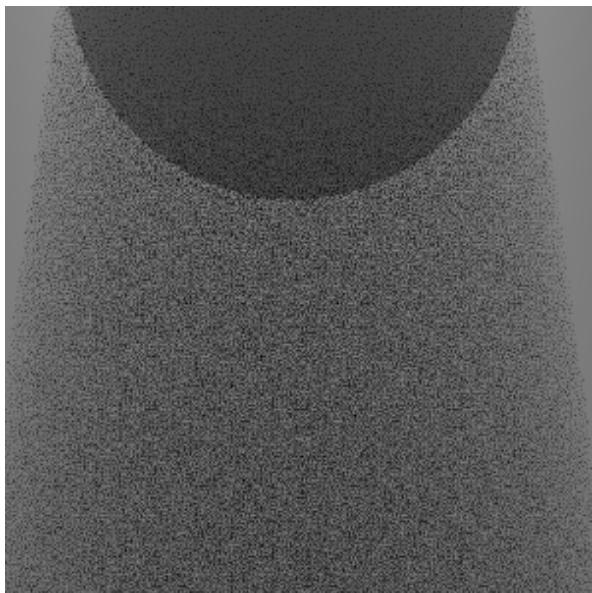


Specular Roughness: 0.2

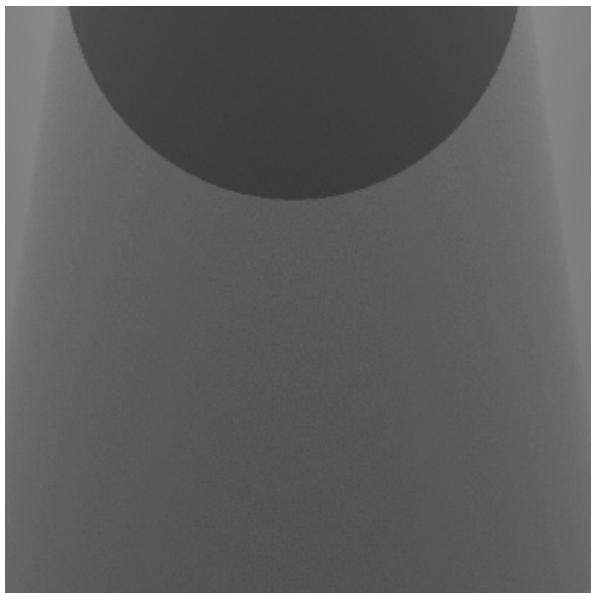


Specular Roughness: 0.5

#### Atmosphere Volume Noise



All samples: 1



Atmosphere Volume samples: 10

*Atmosphere Volume* noise will occur within the shadowed regions of a beam of light. This noise is caused by a lack of samples in the *Atmosphere Volume* shader. The samples are distributed according to the volume density. More samples will refine the quality of the solution.

#### Fireflies/Sparks

Fireflies (bright 'spike' type noise) are typically generated from the reflection of a strong light on a shiny, low-roughness glossy surface. Usually, hundreds of samples participate in a final pixel color. If one of those is a high-valued specular ray (coming from the reflection of a strong light), then there will be many samples with low values and a single sample with a value in the thousands. That single sample will make the whole pixel become white (a "firefly").

#### Clamping

Sample clamping can help eliminate this kind of noise. If you clamp high sample values, a single stray sample will now be diluted and not affect the final color as much. However, this will affect the final dynamic range of the render. Use with caution.

#### Light Decay Filter

Fireflies can also be caused by indirect diffuse illumination from very bright hotspots, in particular when the light sources are very close (or even touch) nearby geometry, such as a light's cabinet or enclosure. The inverse square decay in the light will make those nearby surfaces extremely bright. A workaround is to add a **light decay filter** with a very low ***near start*** value to avoid samples very close to the light. Again, this should be used with caution.

### Other Considerations

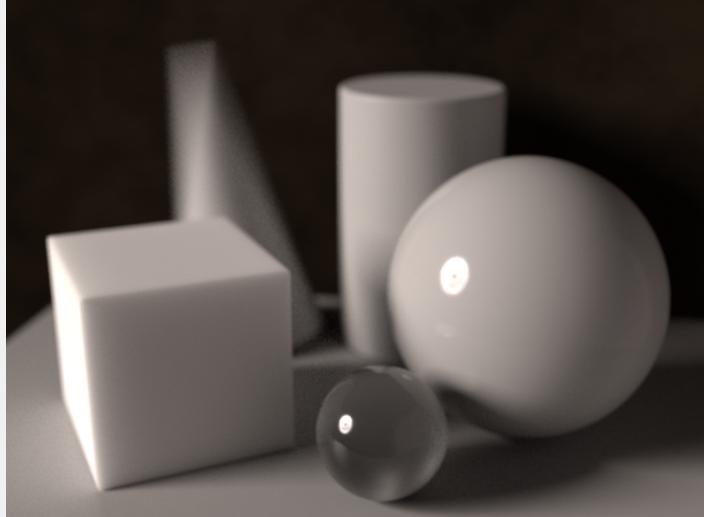
- Noise could come from things not visible in the render (behind the camera).
- Noise could be caused/exacerbated by non-energy conserving shaders, networks or settings.
- Another method to reduce noise is to remove the cause and fake it with special lights. For example, a character whose face is lit by bounce light only: it will be much less noisy to add a specific bounce light.
- Arnold can remove noise easily when it knows where lights are, by sampling the lights directly, but will have problems if there are bright "directional" patches not tagged as lights that contribute significantly to the scene's lighting.

### Clean Renders

#### No DOF

<ul style="list-style-type: none"><li>• <b>Camera (AA) samples:</b> 7</li><li>• <b>Diffuse samples:</b> 3</li><li>• <b>Specular samples:</b> 2</li><li>• <b>Transmission samples:</b> 6</li><li>• <b>SSS samples:</b> 4</li><li>• <b>Light samples:</b> 3</li></ul>	
---	---

#### With DOF

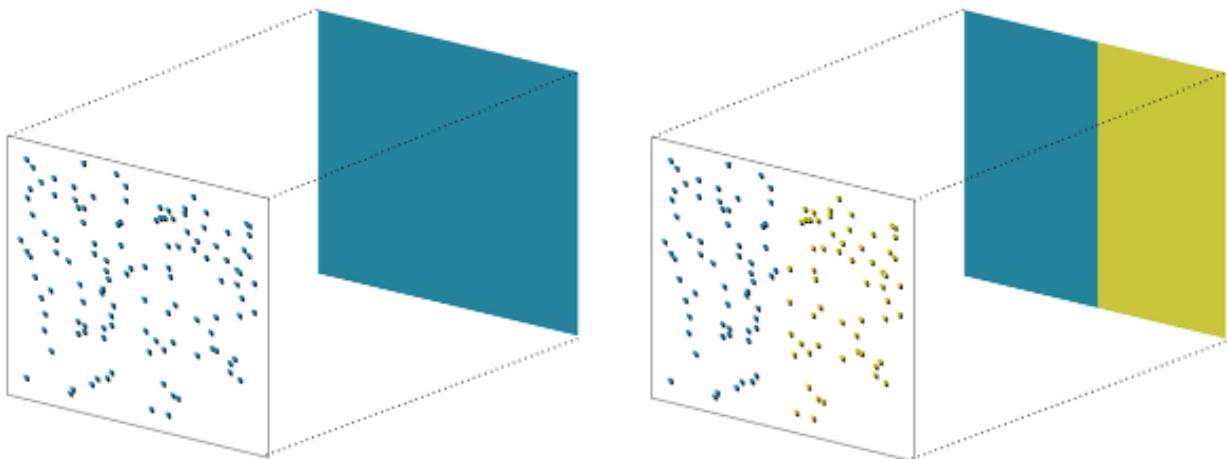
<ul style="list-style-type: none"><li>• <b>Camera (AA) samples:</b> 15</li><li>• <b>Diffuse samples:</b> 1</li><li>• <b>Specular samples:</b> 1</li><li>• <b>Transmission samples:</b> 4</li><li>• <b>SSS samples:</b> 3</li><li>• <b>Light samples:</b> 3</li></ul>	
--	--

When trying to identify noise in your renders, it is useful to render and view AOVs. This enables you to isolate the type of noise and adjust the relevant samples. A general guide to identifying and prioritizing the order in which common noise types can be found in the table below. Note that this is a general guide and that every scene is different.

Noise Visible in	Samples to Adjust
Alpha channel	Camera (AA) samples
Indirect Diffuse	Diffuse samples
Direct Specular (specular noise)	Light samples
Direct Diffuse (shadow noise)	Light samples
Indirect Specular	Specular samples
Transmission	Transmission samples
SSS (direct and indirect)	SSS (direct and indirect) samples
Volume	Volume samples (note that there are also volume samples in lights too)

## Removing Noise Workflow

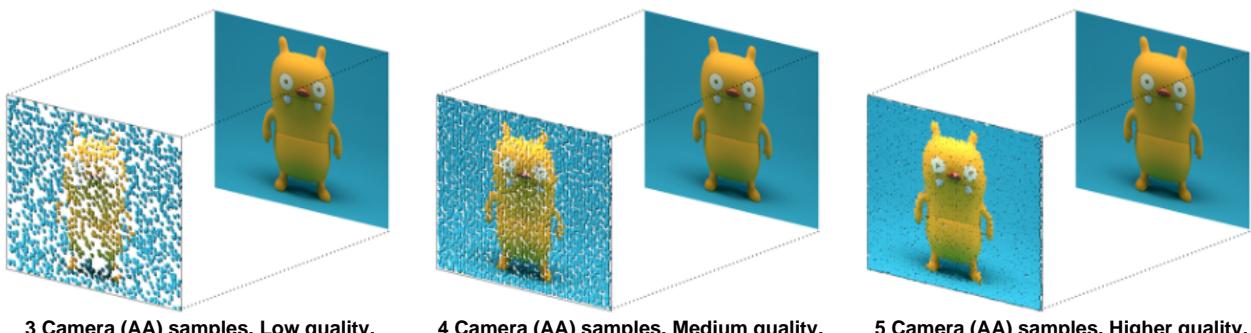
If you have a scene that consists of a simple color and you want to know the average color of the photo, it is easy to see what the color is at any one point by using a low number of samples and then averaging the result.



If the scene is all one color, it only requires a few samples to get the correct average

If the scene is half one color and half another, more samples will still give you a good average

However, if you have a more complex scene (with a character for example), and use only a few samples, sometimes half of them will be yellow and half of them blue. Sometimes all the samples will be one color or the other. Sometimes, by coincidence, all of the samples will be on the yellow character, despite only a part of the scene being the yellow character. A way to improve the average is by increasing the number of Camera (AA) samples.



3 Camera (AA) samples. Low quality.

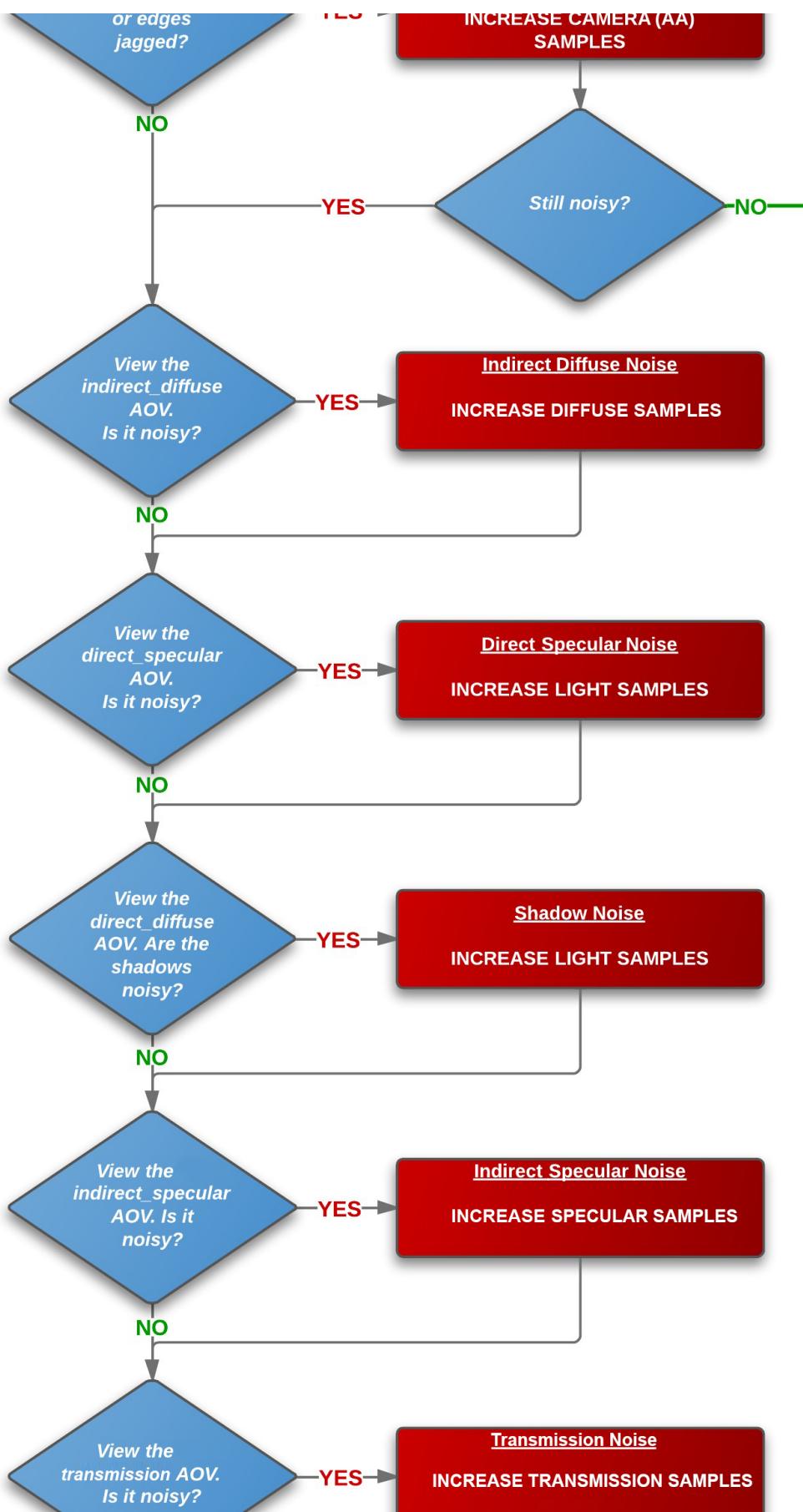
4 Camera (AA) samples. Medium quality.

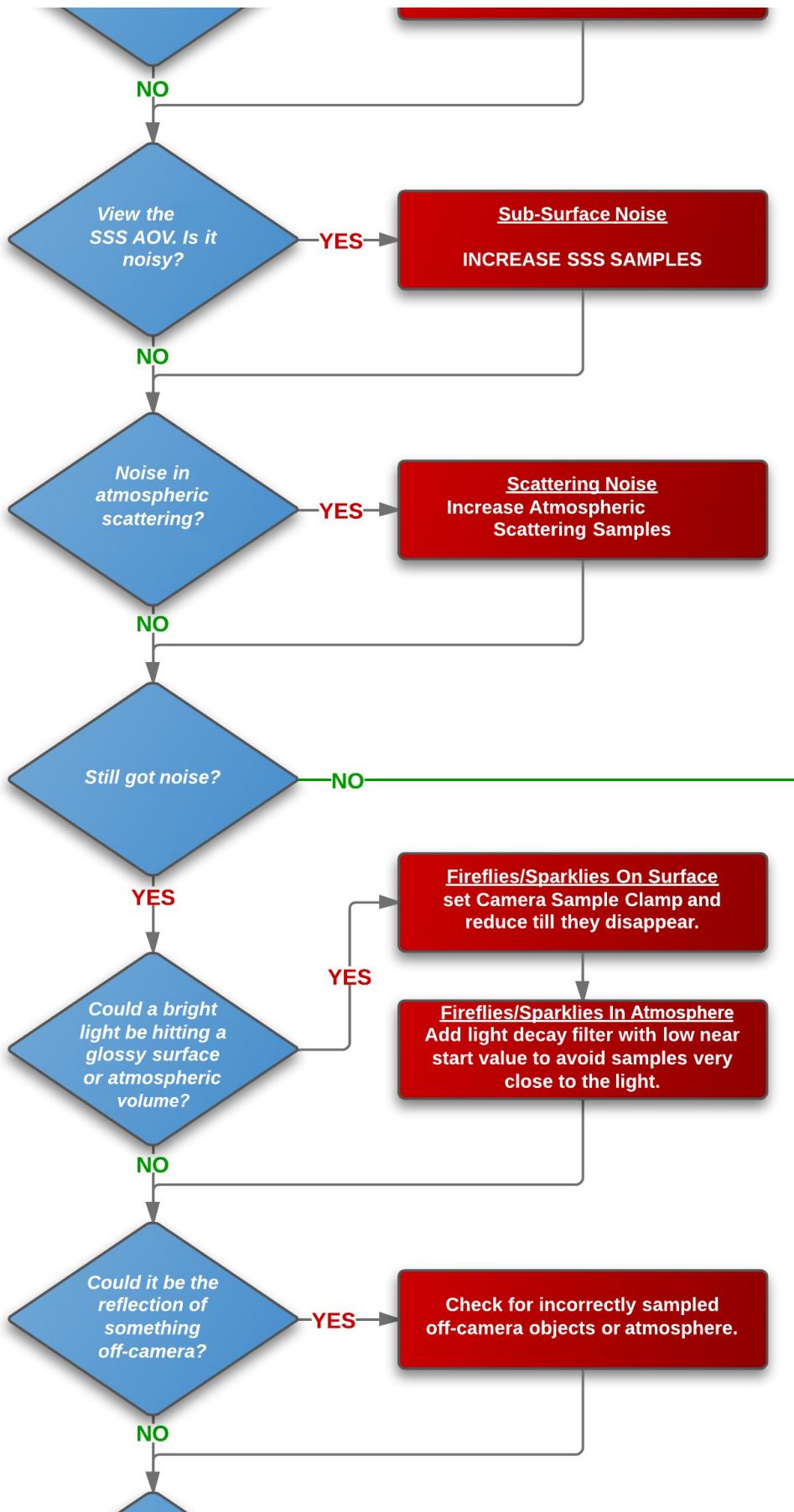
5 Camera (AA) samples. Higher quality.

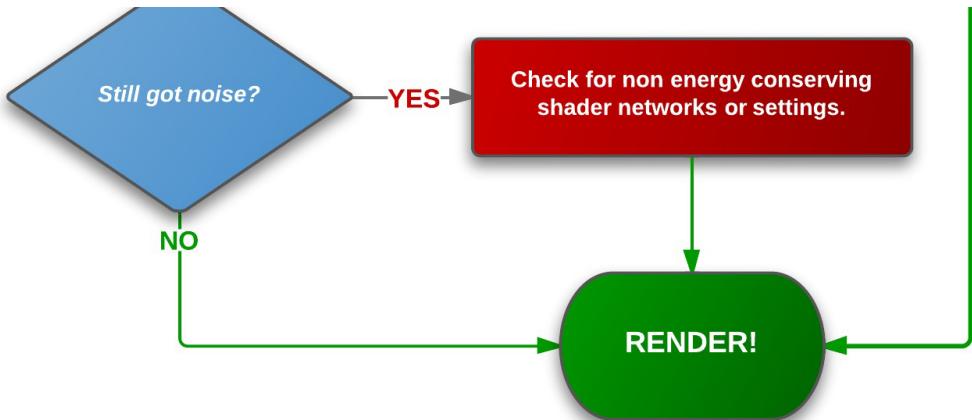
Increasing the number of Camera (AA) samples resolves the noise and improves the quality of the image.

The diagram below shows a good method for identifying the type of noise in your scene and which sampling parameter to use to improve it.





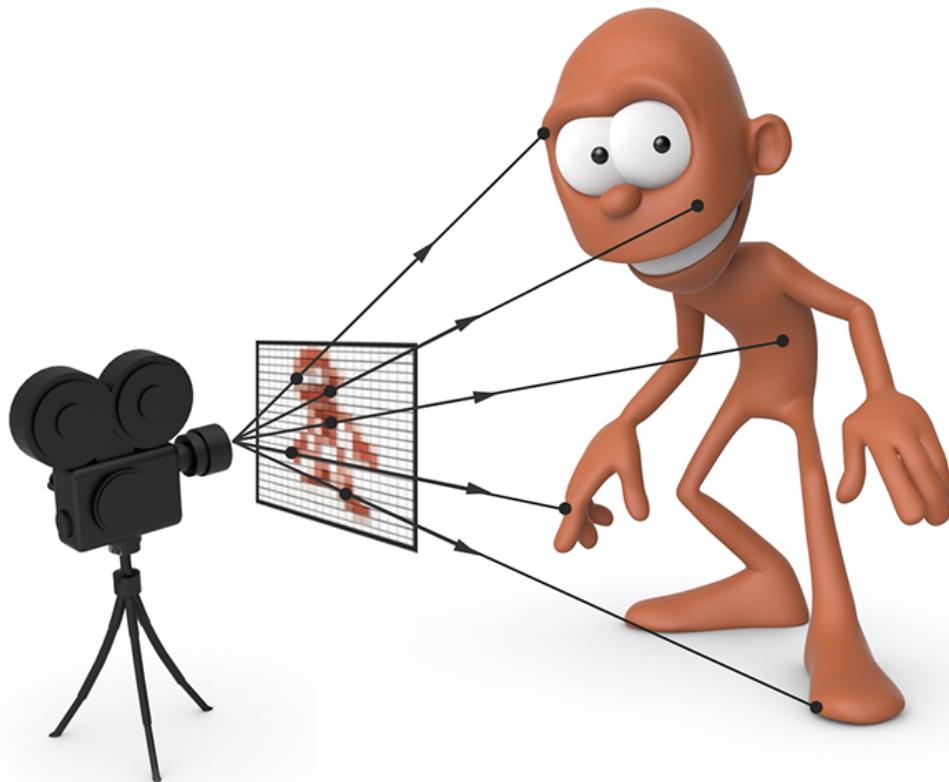




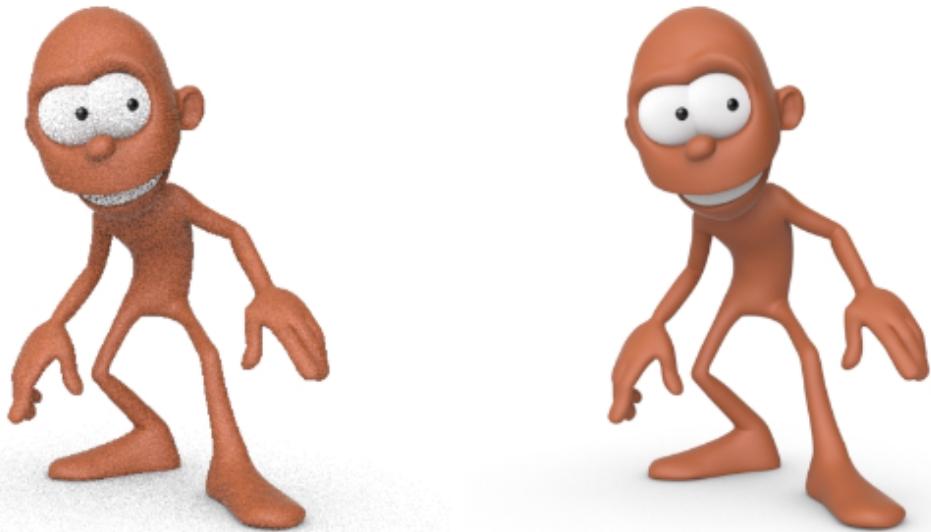
## What is Sampling?

When working with a renderer like [Arnold](#), it is useful to have a basic understanding of the core principles behind ray-trace rendering. Producing a realistic-looking image of a virtual scene requires simulating the propagation of light in the scene all the way from the light sources to the camera. To determine the color of each image pixel, Arnold collects information from the scene geometry, shaders, lights, etc., and traces a number of random light transport paths that connect the objects seen through the pixel to the light sources - a process called 'sampling'. The quality of the resulting image largely depends on the number of paths, or samples, generated for each pixel.

When rendering an image, Arnold must determine a color value for each pixel by examining the scene. Arnold achieves this by sending out a number of rays from the camera's position until they hit an object in the scene. Every time a ray hits an object, it will perform some calculations that will eventually return a piece of information about the object (its color, for instance). This process can basically be described as 'sampling' the pixels in the virtual camera's image plane.



Renders computed with too few samples in each pixel can be noisy. Increasing the number of samples per pixel gives a less noisy image which better represents the actual scene. The images below show a comparison when increasing the number of Camera (AA) samples in a scene. Not only does increasing Camera (AA) samples reduce aliasing noise, but it also adds more secondary ray samples which also reduces noise in the lighting.



**1 Camera (AA): 1x1=1 sample per pixel (image appears noisy)**

**6 Camera (AA): 6x6=36 (noise is reduced)**

### ***Camera (AA) Sampling***

So, what is camera sampling? Basically, a number of rays are shot from the camera through each required pixel of the rendered screen window into the scene. These are called 'primary rays', but sometimes they are referred to as 'eye' or 'camera rays' (Camera (AA) (anti-aliasing)) as they are cast from the rendering view. Sometimes they are called 'pixel samples'.

The Camera (AA) value controls the pixel supersampling rate or the number of rays per pixel that will be traced from the camera. The higher the number of samples, the better the anti-aliasing quality, but the longer the render times.

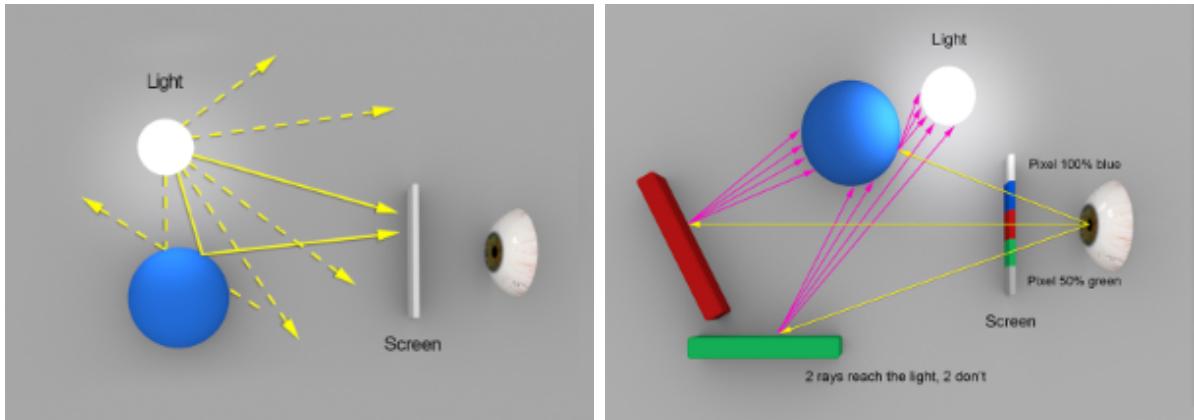
Note that this process is not linear, as, for each of these sampling rates, the actual number of samples taken is the square of the input value. For example, if Camera (AA) samples is 3, it means that  $3 \times 3 = 9$  samples will be used for anti-aliasing. If Diffuse samples is 2, then  $2 \times 2 = 4$  samples will be used for the Diffuse GI. The same applies to the other values.

These AA samples could be described as primary rays (or pixel samples) and determine the overall quality of the image being rendered. Increasing the AA samples improves the overall quality of the image, but it can often be wasteful because it increases the sampling rate for everything. When the source of noise is identified in the scene, it is more efficient to set the sampling values to increase the number of only those specific rays. If for example, there is a scene with lots of motion blur, a larger Camera (AA) sample is required, and therefore the other sample values can be reduced. Likewise, increasing Camera (AA) can also benefit indirect lighting and therefore not as many Diffuse samples would be required. However, if you were to reduce noise on the skin of a character, it would be more efficient to have higher SSS samples and not such a high value for Camera (AA) samples.

### ***Light Sampling***

Direct lighting rays can be described as rays which deal with lights. These rays travel from a position in the scene toward the various light sources. These rays determine if a surface is in shadow, and if not, lighting information can be calculated.

Noise from lights can sometimes be difficult to diagnose, particularly if it is a light with a large area or size. In these cases, it can sometimes be mistaken for indirect diffuse noise. This highlights the necessity for testing different noise ray types. If the issue is shadow noise, then we can simply toggle ignore shadows in the Arnold render settings, and the noise will completely resolve. The images below show how a light is traced in Arnold.



Some light rays hit an object while others do not

Light sampling in Arnold

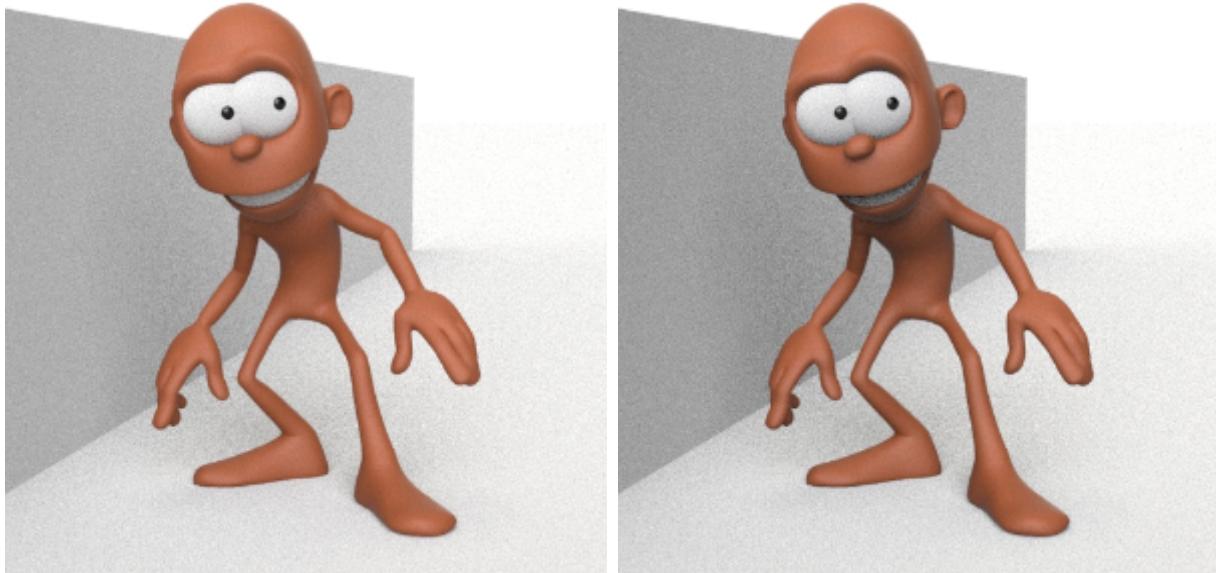
The same noise problems described earlier can occur with direct and indirect rays. Noise from direct light sources usually appear in specular highlights or in large, soft shadows from area lights. If this is the case, the number of light samples needs to be increased. The images below show the difference in the shadow noise when increasing the number of light samples from 1 to 3.



1 light sample. Very noisy.

3 light samples. Noise is improved.

It can be useful to separate one source of noise from another, when trying to evaluate noise in a scene. Disabling 'Diffuse' rays (0) can help identify which type of noise is present. When attempting to optimize the number of indirect diffuse rays in a scene, rather than increasing the number of Diffuse rays (render time will double for each Diffuse sample), the source of the noise will need to be isolated first.



Both light noise and Diffuse noise (difficult to tell which is which).

Diffuse set to 0 (disabled). Light noise is easier to identify.

### *Diffuse Ray Sampling*



Indirect (diffuse) lighting from emissive green cube. Rollover image to view direct lighting only (Diffuse Samples 0)

Indirect Diffuse rays are rays that interact with objects and their surface shaders. Therefore, rays travel in the scene in directions determined by the shader assigned to the object. Transmission rays travel through objects. When a ray hits a reflective object, such as a mirror, only one reflection ray is traced from that point in a direction determined by the incoming ray. In contrast, diffuse rays are sampled randomly over the hemisphere around the hit point.

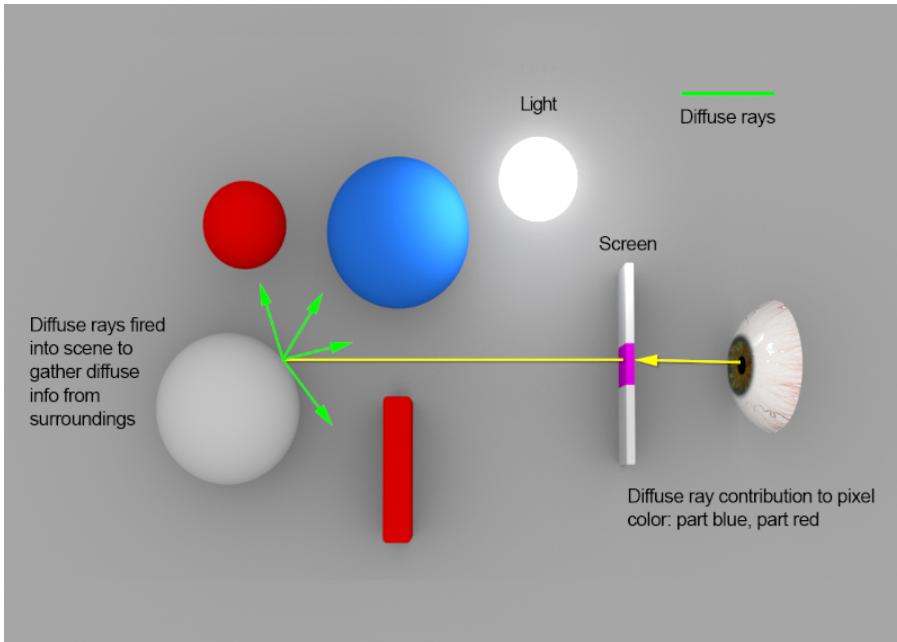
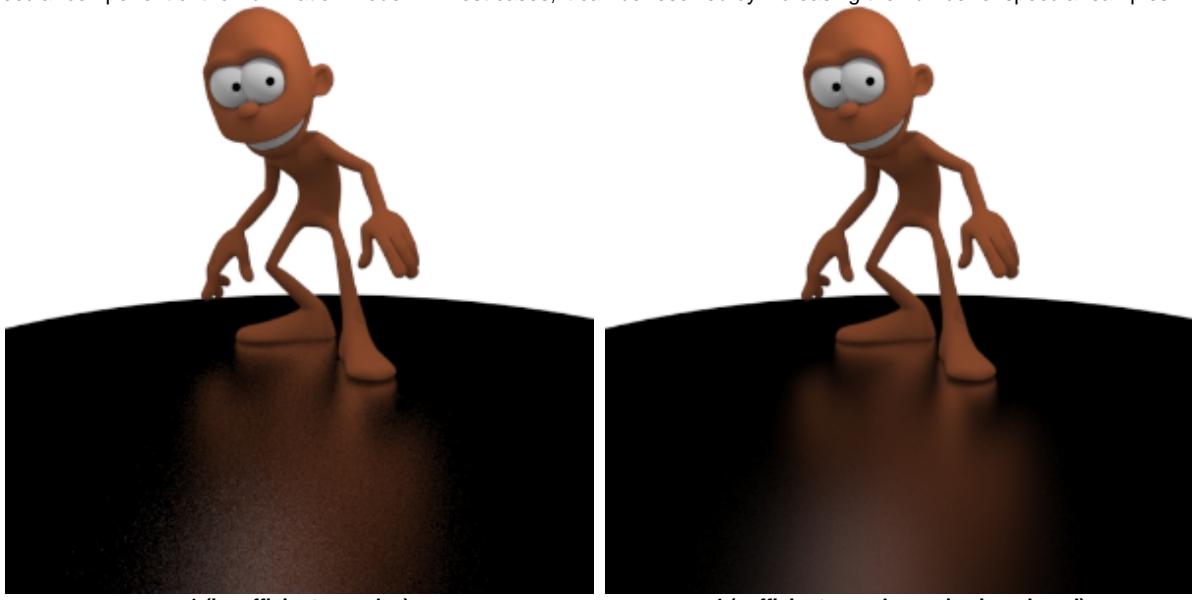


Diagram showing how diffuse rays are propagated in an Arnold render

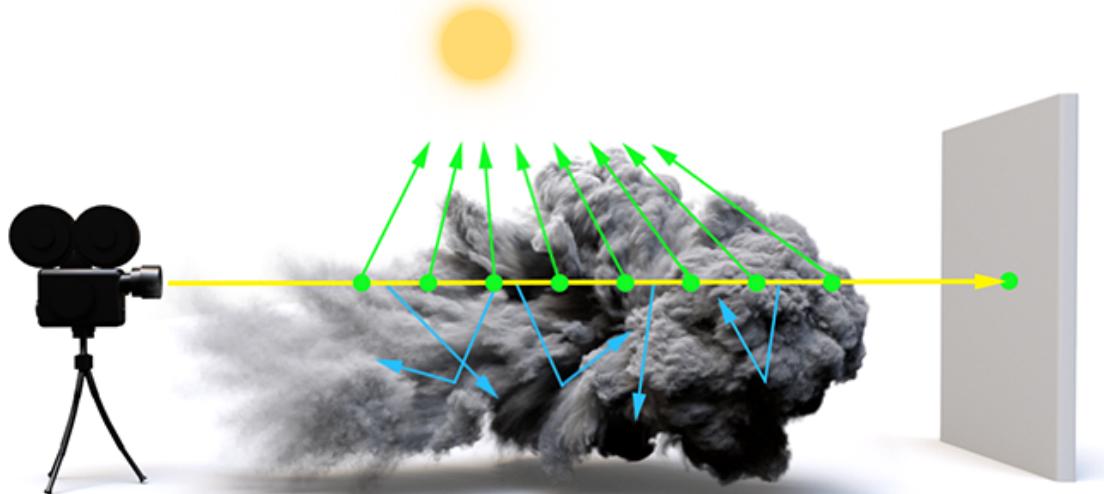
### **Specular Ray Sampling**

When rendering glossy specular surfaces, noise can also be a problem, as with diffuse rays. Bright hotspots can cause noise in indirect specular samples. For example, wide specular reflections of small, bright lights. If the issue is noise in a specular highlight, it needs to be confirmed that the source is the direct light and not a secondary ray type (such as specular). This is easy to achieve by setting the `GI_diffuse_depth`, `GI_specular_depth` to zero (this essentially turns off all global illumination). If the noise is still there, we know it is the direct specular component of the illumination model. In most cases, it can be resolved by increasing the number of specular samples.



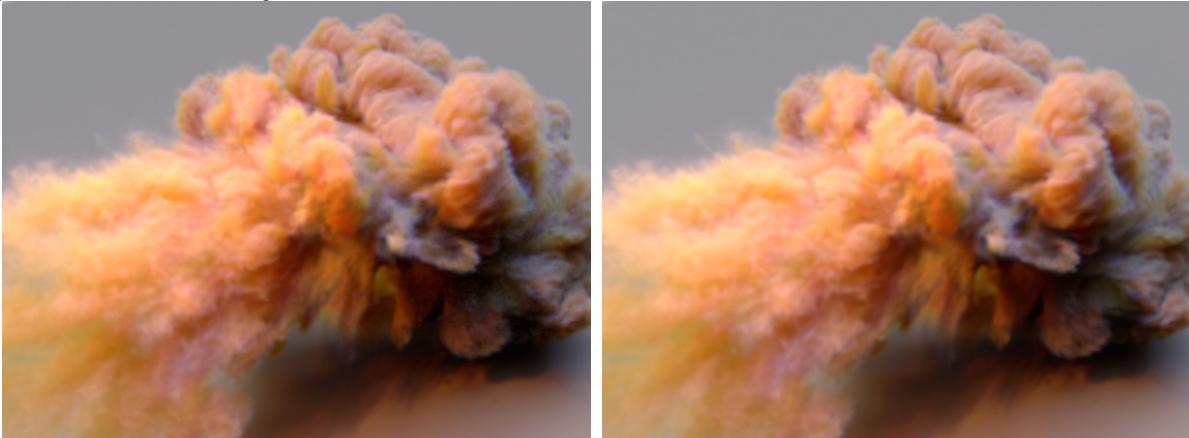
### **Volumes**

When sampling volumes, direct (AA) rays sample the volume multiple times as they travel through it. Indirect rays (Volume Indirect) behave similarly, sent multiple times as the ray 'steps' (volume step size) through the volumetric object. Therefore, sampling a volume usually takes longer to calculate than sampling a surface.



Volume sampling. Direct lighting (green). Indirect lighting (blue).

At each step the ray travels, it evaluates the shader and accumulates the density of the volume. When these density values become more erratic throughout the volume, nearby rays can calculate substantially different values, introducing noise into the render. In this case, it may be necessary to render using more light volume samples or to reduce the step size in the volume. Even with low sampling values, it can be expensive to render clean images with volumes.



There are three things to consider when rendering volumes; ray marching, direct light, and indirect light sampling, all of which have different sampling settings.

#### **Step Size**

The `step_size` of the ray marching process. Too low a sampling rate here will cause the volume rendering to miss small details in the volume, making them noisy, making them thinner than they should be, or making a bad estimate of the exact position of the 'surface' of a 'solid' volume like a pyroclastic cloud. A bad step size has a snowball effect that can make the rest of the sampling process noisier. This is pretty easy to test by looking at the alpha channel, or by adding some emission to the volume and turning off all lighting. If the emission/alpha channel is noisy at the desired AA setting, then the `step_size` is probably too large.



Step size correct

Artifacts appear in the alpha when the step size is too large

Step size is actually in object-space and not in world-space. This is so that the sampling quality remains the same upon scaling of the volume's transformation. However, unreasonably large step sizes should be avoided. Otherwise, the volume will appear thinned and washed out. In the example images below the volume has been scaled up to 100 units to exaggerate this effect (1/25/50 are relative to the size of this volume).



1

25

50

Generally, the step size needs to be as large as possible before visibly noticeable artifacts occur. Low Step Sizes will increase render times. For example, when the step size is 0.1, and the volume is 10 units big in world space, there are approximately 100 primary samples, and therefore the volume shader is called 100 times.

#### ***Direct Lighting***

When the step size is correct, noise can come from either direct or indirect lighting. Direct lighting is pretty easy to detect. The indirect light needs to be toggled off, and if the render is noisy, each light needs to be rendered one at a time until the light that is responsible for the noise is found, and turn up their `volume_samples`. If the direct lighting noise doesn't significantly decrease as `volume_samples` increases, then it's probably due to a too-low `step_size` or simply difficult to sample light interactions (highly anisotropic volumes, heavily textured mesh\_lights, quadratic falloff, etc.). In this case, there's not much to do except to eliminate the samples via clamping or filtering.

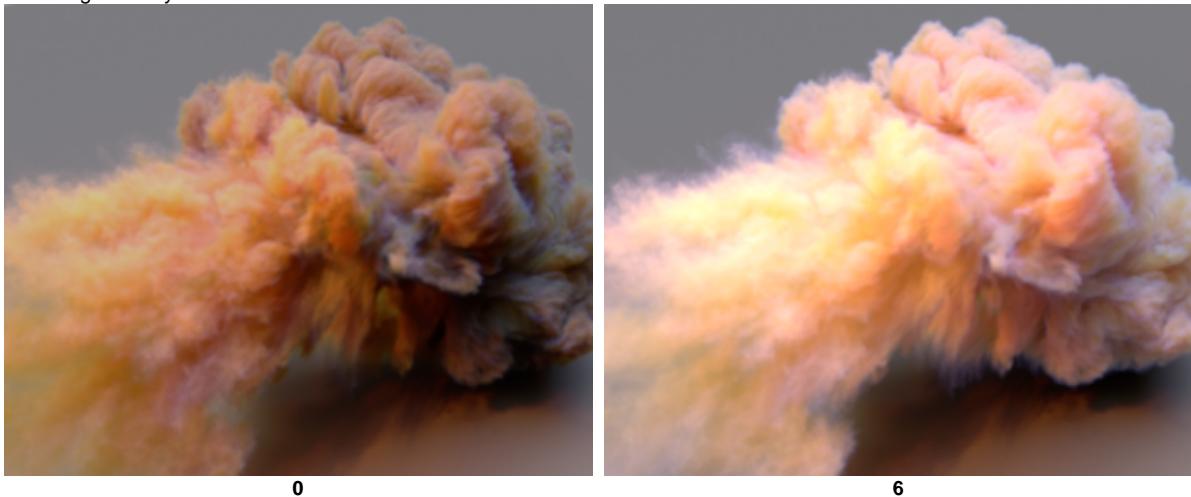
#### ***Indirect Lighting***

Indirect noise is also easy to identify. `GI_volume_depth` of a noisy volume needs to be disabled, and if the render is no longer noisy, then the problem is indirect sampling. If the quality needs to be improved, the number of `volume_indirect_samples` needs to be increased. If

increasing this doesn't reduce noise, then the noise could again be from difficult to sample light interactions (multiple scattering, anisotropic volumes, lights embedded in the volume, etc.) and as mentioned before there's not much to do except to eliminate the samples via clamping or filtering.

#### **Volume ray depth**

Increasing the volume ray depth can make a dramatic difference to the appearance of a volume. However, be aware that increasing the volume ray depth will increase the number of multiple scattering bounces within the volume (defaults to 0) and therefore render times will increase significantly.



*Pepe model by Daniel M. Lara ([Pepeland](#)).*

# Shaders

- AOVs
- BSDFs
- Closures
  - Converting to Closures
- Light Filters
- Light Loops
- Light Path Expression AOVs
- OSL Shaders
- Shader Data Type Conversions
- Trace Sets
- Vector Displacement
- Volume Shaders

## AOVs

Arbitrary output variables (AOVs) make it possible to output image besides the beauty render, for using in compositing software.

### Light Path Expression AOVs

For Arnold light path expressions are used to output light into specific AOVs, with the beauty RGBA AOV being the most commonly used one. Rather than writing to specific AOVs from shaders, LPEs can be used to extract specific light paths using regular expressions. Built-in LPEs are available for the common cases.

See [Light Path Expression AOVs](#) for more details.

### Builtin AOVs

In addition to builtin LPEs, these builtin AOVs are available.

Name	Type	Description
A	FLOAT	Alpha channel
Z	FLOAT	Z depth
opacity	RGB	Per RGB channel alpha
volume_opacity	RGB	Per RGB channel alpha, using volumes only
ID	UINT	id parameter from object
P	VECTOR	World space position
Pref	VECTOR	Reference space position, from Pref user data
N	VECTOR	Normal
motionvector	VECTOR2	Screen space motion vector
shadow_matte	RGBA	Shadow
cputime	FLOAT	Time taken to render pixel
raycount	FLOAT	Number of rays traced to render pixel
shader	NODE	Pointer to shader node, for filters and drivers
object	NODE	Pointer to object node, for filters and drivers

### Custom AOVs

Shaders can output custom AOVs for cases that are not covered by LPEs and builtin AOVs. For example to output mattes or masks, ambient occlusion, textures, etc. AOVs are only written for camera rays.

Here's an example shader, writing a noise texture to a float AOV:

```
write_aov.cpp

#include <ai.h>

AT_SHADER_NODE_EXPORT_METHODS(WriteAovMtd);

struct WriteAovData
{
    AtString aov_name;
};

enum WriteAovParams
```

```

{
    p_aov_name,
};

node_parameters
{
    AiParameterStr("aov_name", " ");
    AiMetaDataSetBool(nentry, "aov_name", "linkable", false);
}

node_initialize
{
    AiNodeSetLocalData(node, new WriteAovData());
}

node_update
{
    // register AOV
    WriteAovData *data = (WriteAovData*)AiNodeGetLocalData(node);
    data->aov_name = AiNodeGetStr(node, "aov_name");
    AiAOVRegister(data->aov_name, AI_TYPE_FLOAT, AI_AOV_BLEND_OPACITY);
}

node_finish
{
    WriteAovData *data = (WriteAovData*)AiNodeGetLocalData(node);
    delete data;
}

shader_evaluate
{
    const WriteAovData *data = (WriteAovData*)AiNodeGetLocalData(node);
    const float noise = AiPerlin3(sg->P);

    // write AOV only if in use
    if ((sg->Rt & AI_RAY_CAMERA) && AiAOVEnabled(data->aov_name, AI_TYPE_FLOAT))
        AiAOVSetFlt(sg, data->aov_name, noise);

    sg->out.FLT() = noise;
}

node_loader
{
    if (i != 0)
        return false;
    node->methods      = WriteAovMtd;
    node->output_type  = AI_TYPE_FLOAT;
    node->name         = "write_aov";
    node->node_type    = AI_NODE_SHADER;
}

```

```
    strcpy(node->version, AI_VERSION);
    return true;
}
```

## BSDFs

In Arnold 5.0, the BSDF API has been redesigned to support more advanced rendering algorithms, more advanced BSDFs, and more optimized implementations.

### Diffuse Example

Here is a simple diffuse BSDF example, with a shader that uses the BSDF to integrate direct and indirect light.

#### diffuse\_bsdf.cpp

[Expand source](#)

```
#include "diffuse_bsdf.h"

struct DiffuseBSDF
{
    /* parameters */
    AtVector N;
    /* set in bsdf_init */
    AtVector Ng, Ns;
};

AI_BSDF_EXPORT_METHODS(DiffuseBSDFMtd);

bsdf_init
{
    DiffuseBSDF *data = (DiffuseBSDF*)AiBSDFGetData(bsdf);

    // store forward facing smooth normal for bump shadowing
    data->Ns = (sg->Ngf == sg->Ng) ? sg->Ns : -sg->Ns;

    // store geometric normal to clip samples below the surface
    data->Ng = sg->Ngf;

    // initialize the BSDF lobes. in this case we just have a single
    // diffuse lobe with no specific flags or label
    static const AtBSDFLobeInfo lobe_info[1] = {{AI_RAY_DIFFUSE_REFLECT, 0,
AtString()}};
    AiBSDFInitLobes(bsdf, lobe_info, 1);

    // specify that we will only reflect light in the hemisphere around N
    AiBSDFInitNormal(bsdf, data->N, true);
}

bsdf_sample
{
    DiffuseBSDF *data = (DiffuseBSDF*)AiBSDFGetData(bsdf);

    // sample cosine weighted incoming light direction
    AtVector U, V;
    AiV3BuildLocalFrame(U, V, data->N);
    float sin_theta = sqrtf(rnd.x);
    float phi = 2 * AI_PI * rnd.y;
    float cosNI = sqrtf(1 - rnd.x);
    AtVector wi = sin_theta * cosf(phi) * U +
                  sin_theta * sinf(phi) * V +
                  cosNI * N;
```

```

cosNI * data->N;

// discard rays below the hemisphere
if (!(AiV3Dot(wi, data->Ng) > 0))
    return AI_BSDF_LOBE_MASK_NONE;

// since we have perfect importance sampling, the weight (BRDF / pdf) is 1
// except for the bump shadowing, which is used to avoid artifacts when the
// shading normal differs significantly from the smooth surface normal
const float weight = AiBSDFBumpShadow(data->Ns, data->N, wi);

// pdf for cosine weighted importance sampling
const float pdf = cosNI * AI_ONEOVERPI;

// return output direction vectors, we don't compute differentials here
out_wi = AtVectorDv(wi);

// specify that we sampled the first (and only) lobe
out_lobe_index = 0;

// return weight and pdf
out_lobes[0] = AtBSDFLobeSample(AtRGB(weight), 0.0f, pdf);

// indicate that we have valid lobe samples for all the requested lobes,
// which is just one lobe in this case
return lobe_mask;
}

bsdf_eval
{
    DiffuseBSDF *data = (DiffuseBSDF*)AiBSDFGetData(bsdf);

    // discard rays below the hemisphere
    const float cosNI = AiV3Dot(data->N, wi);
    if (cosNI <= 0.f)
        return AI_BSDF_LOBE_MASK_NONE;

    // return weight and pdf, same as in bsdf_sample
    const float weight = AiBSDFBumpShadow(data->Ns, data->N, wi);
    const float pdf = cosNI * AI_ONEOVERPI;
    out_lobes[0] = AtBSDFLobeSample(AtRGB(weight), 0.0f, pdf);

    return lobe_mask;
}

AtBSDF* DiffuseBSDFCreate(const AtShaderGlobals* sg, const AtRGB& weight, const
                           AtVector& N)
{
    AtBSDF* bsdf = AiBSDF(sg, weight, DiffuseBSDFMtd, sizeof(DiffuseBSDF));
    DiffuseBSDF* data = (DiffuseBSDF*)AiBSDFGetData(bsdf);
}

```

```
    data->N = N;
    return bsdf;
}
```

### diffuse\_bsdf.h

[Expand source](#)

```
#pragma once

#include <ai_shader_bsdf.h>
#include <ai_shaderglobals.h>

AtBSDF* DiffuseBSDFCreate(const AtShaderGlobals* sg, const AtRGB& weight, const
AtVector& N);
```

## diffuse\_shader.cpp

[Expand source](#)

```
#include "diffuse_bsdf.h"

#include <ai.h>

AI_SHADER_NODE_EXPORT_METHODS(DiffuseMtd)

enum DiffuseParams {
    p_color,
};

node_parameters
{
    AiParameterRGB("color", 0.8f, 0.8f, 0.8f);
}

node_initialize
{
}

node_update
{
}

node_finish
{
}

shader_evaluate
{
    // early out for shadow rays and black color
    if (sg->Rt & AI_RAY_SHADOW)
        return;

    AtRGB color = AiShaderEvalParamRGB(p_color);
    if (AiColorIsSmall(color))
        return;

    sg->out.CLOSURE() = DiffuseBSDFCreate(sg, color, sg->Nf);
}

node_loader
{
    if (i>0)
        return false;

    node->methods      = DiffuseMtd;
    node->output_type  = AI_TYPE_CLOSURE;
    node->name         = "diffuse";
    node->node_type    = AI_NODE_SHADER;
    strcpy(node->version, AI_VERSION);
    return true;
}
```

## Methods

A BSDF consists of a struct to store its parameters, and a number of methods. A new BSDF is created using `AiBSDF`, which receives a method table and allocates a struct to store parameters. After shader evaluation is finished, the integrator will initialize, evaluate and sample the BSDF.

- `bsdf_init`: Called before the BSDF is evaluated or sampled for the first time. BSDF initialization can include: ensuring that the provided parameters are within a valid range, storing local geometry data for later evaluation and sampling (geometric normal, outgoing view direction, ...), and precomputing any data needed for evaluation and sampling. Here the BSDF must also provide information about its lobes, and optionally provide bounds for more efficient light culling.
- `bsdf_eval`: Evaluates the BSDF for a given incoming light direction and the current outgoing view direction. If the BSDF consists of multiple lobes, `lobe_mask` describes which lobes must be evaluated. The result of this evaluation for each lobe is:
  - RGB `weight`, defined as `BSDF * cos(N.wi) / pdf`. The cosine of the angle between the surface normal and the incoming light direction must be included, and the weight is divided by the probability density. For a BSDF that provides perfect importance sampling, this weight would be 1.
  - `pdf`, the probability density for sampling the the incoming light direction with `bsdf_sample`.
- `bsdf_sample`: Sample an incoming light direction and evaluate the BSDF for this direction. This function returns:
  - Sampled incoming light direction `wi`.
  - Index of the lobe that was sampled.
  - RGB `weight` and `pdf`, matching `bsdf_eval` for the same incoming light direction.
- `bsdf_interior`: Optionally return a list of volume closures to fill the interior of the volume. The typical example would be a glass BSDF returning a volume absorption closure.

## Lobes

BSDF can consist of multiple lobes, for example multiple layers in a layered BSDF, or separate reflection and refraction components in glass BSDF. Each lobe has an associated ray type and AOV name. This makes it possible to:

- Output each lobe to a separate AOV.
- Independently control the diffuse, glossy and refraction depth and number of samples.
- Let Arnold do more efficient sampling by separating lobes with distinct shapes.

In BSDF initialization the BSDF specifies the number of lobes it consists of, and provides an array with the ray type and AOV name for each.

The evaluation and sample methods then receive a lobe bitmask to indicate which lobes to evaluate or sample respectively. If the sample method receives a lobe mask with multiple lobes, it is up to the BSDF to pick one of the lobes to sample, ideally importance sampling based on how much each lobe contributes.

These methods output an array of lobe samples, and return a lobe bitmask to indicate which lobe samples were filled in. This may be a subset or superset of the lobes specified with the input lobe mask.

## Bounds

If all reflected incoming light on the BSDF is contained in a hemisphere, it is possible to specify the normal of that hemisphere. Providing this information is not strictly required, but can help speed up rendering by letting Arnold more quickly discard light that is outside the bounds of the BSDF.

## Bump Mapping

BSDF may use normals different than the smooth surface normal `Ns`, using bump or normal mapping. If this modified normal is very different than the smooth surface normal, artifacts may appear however. An `AiBSDFBumpShadow` utility function is provided to hide such artifacts by adding extra shadowing as part of BSDF evaluation.

This function takes as input the forward facing smooth normal, the bump mapped normal, and the incoming light direction, and outputs a factor to multiply with the BSDF weight.

## Ray Differentials

Ray directions returned by `bsdf_sample` include ray differentials. For good texture filtering performance and quality it is important to provide ray differentials. The `AiReflectWithDerivs` and `AiRefractWithDerivs` utility functions can be used to compute reflected or refracted vectors with derivatives.

Example for a simple perfectly sharp specular BSDF:

```
AtVectorDv I = AtVectorDv(sg->Rd, sg->dDdx, sg->dDdy);
AtVectorDv N = AtVectorDv(sg->Nf, sg->dNdx, sg->dNdy);
AtVectorDv R = AiReflectWithDerivs(I, N);
```

## Roughness Clamping

Unidirectional path tracers can't resolve caustics efficiently. Sharp specular bounces seen through a rough specular or diffuse bounce are too noisy without any compensation. Built-in BSDFs automatically increase their roughness to reduce caustic noise at the cost of bias. `options.indirect_glossy.blur` controls the amount of blurring, with 0 resulting in unbiased renders.

Custom BSDFs can use the same method, by clamping their roughness with the automatically estimated minimum roughness provided by the `AiBSDFMinRoughness` function.

```
bsdf_init
{
    ...
    // clamp roughness based on path history
    data->roughness = AiMax(AiBSDFMinRoughness(sg), data->roughness);
    ...
}
```

## Exit Colors

Glass shaders often require many bounces to escape. If the number of bounces is limited, this leads to dark patches. Each BSDF lobe has a flag that can be set to use the background color or a fixed white color when the number of bounces has been exceeded.

## Bidirectional Rendering

The API is designed to work with bidirectional rendering algorithms. However as Arnold does not provide such an integrator yet, BSDFs are not yet required to provide implementations compatible with such rendering methods. The `reverse_pdf` member of lobe samples is a placeholder and ignored currently.

## Closures

In Arnold 5.0 surfaces and volume shaders return closures rather than final colors. Closures describe the way surfaces and volumes scatter light, leaving the lights loops and integration to Arnold. This approach makes more optimizations and better rendering algorithms possible.

## Types

### **BSDF and BSSRDF Closures**

BSDF and BSSRDF closures define how light scatters on and below surfaces.

#### **Diffuse BSDF**

```
// OSL
result = 0.8 * diffuse(N);

// C++
sg->out.CLOSURE() = AiOrenNayarBSDF(sg, AtRGB(0.8f), sg->Nf);
```

#### **BSSRDF**

```
// OSL
result = weight * empirical_bssrdf(mean_free_path, albedo);

// C++
sg->out.CLOSURE() = AiClosureEmpiricalBSSRDF(sg, weight, mean_free_path, albedo);
```

### **Refraction BSDFs and Absorption**

For refraction, the BSDF takes an interior closure list parameter that defines the interior of the object. This can be used to model volume absorption or scattering inside glass for example.

#### **BSSRDF**

```
// C++
AtClosureList interior = AiClosureVolumeAbsorption(sg, absorption_coefficient);
sg->out.CLOSURE() = AiMicrofacetRefractionBSDF(sg, ..., interior);
```

### **Emission Closure**

The emission closure is used to emit light from surfaces.

## Emission

```
// OSL
result = intensity * color * emission();

// C++
sg->out.CLOSURE() = AiClosureEmission(sg, intensity * color);
```

## Transparent and Matte Closures

Surfaces are opaque by default, and the transparent and matte closures can be used to make them transparent or to affect the alpha channel. When making a surface transparent or matte, other surface closures should have their weight reduced accordingly, so that the total weight of all closures does not exceed 1 and energy is conserved. Mixing with other closures can be done as follows:

### Diffuse BSDF with transparency

```
// OSL
result = opacity * 0.8 * diffuse(N) + (1 - opacity) * transparent();

// C++
AtClosureList closures;
closures.add(AiOrenNayarBSDF(sg, opacity * AtRGB(0.8f), sg->Nf));
closures.add(AiClosureTransparent(sg, 1 - opacity));
sg->out.CLOSURE() = closures;
```

The transparent closure makes the surface transparent, revealing objects behind it. The matte closure creates a hole in the alpha channel, while blocking objects behind the surface. This can be used to composite other objects into the image after rendering.

The relation to opacity and alpha is as follows:

```
opacity = 1 - transparent
alpha = 1 - transparent - matte
```

## Stochastic Transparency

For best performance, stochastic opacity should be used. For OSL shaders this is applied automatically, for C++ it can be done like this. Note that for shadow rays only opacity is needed and creating BSDF and evaluating any parameters needed for them should be skipped for best performance.

## Stochastic transparency

```
// handle opacity
AtRGB opacity = AiShaderGlobalsStochasticOpacity(sg,
AiShaderEvalParamOpacity(p_opacity));
if (opacity != AI_RGB_WHITE)
{
    sg->out.CLOSURE() = AiClosureTransparent(sg, AI_RGB_WHITE - opacity);
    // early out for nearly fully transparent objects
    if (AiAll(opacity < AI_OPACITY_EPSILON))
        return;
}

// early out for shadow rays
if (sg->Rt & AI_RAY_SHADOW)
    return;

// create shader closures
AtClosureList closures = ...;

// write closures
if (opacity != AI_RGB_WHITE)
{
    closures *= opacity;
    sg->out.CLOSURE().add(closures);
}
else
{
    sg->out.CLOSURE() = closures;
}
```

## Volume Closures

The weights of volume closures are absorption, scattering and volume coefficients, with higher values resulting in denser volumes. Shadow rays only need absorption, and so may skip computation of scattering and emission and avoid evaluating any linked parameters needed for them.

## Volume

```
// OSL
if (raytype("shadow"))
    result = density * volume_absorption();
else
    result = density * volume_henyey_greenstein(absorption, scattering, emission,
anisotropy);

// C++
if (sg->Rt & AI_RAY_SHADOW)
    sg->out.CLOSURE() = AiClosureVolumeAbsorption(sg, AtRGB(density));
else
    sg->out.CLOSURE() = AiClosureVolumeHenyeyGreenstein(sg, density * (1 -
scattering),
                                                       density * scattering,
                                                       density * emission,
                                                       anisotropy);
```

## AOVs

Shaders using closures cannot write lighting to AOVs themselves, rather [Light Path Expression AOVs](#) can be used.

## Converting to Closures

Examples of converting Arnold 4 shaders to use [closures](#).

### Diffuse Shader

```
// old
AtRGB direct_diffuse = Kd * AiDirectDiffuse(sg->Nf, sg);
AtRGB indirect_diffuse = Kd * AiIndirectDiffuse(sg->Nf, sg);
sg->out.RGB = direct_diffuse + indirect_diffuse;
AiaOVSetRGB(sg, "direct_diffuse", direct_diffuse);
AiaOVSetRGB(sg, "indirect_diffuse", indirect_diffuse);

// new, light path expressions AOVs are automatically written by closures
sg->out.CLOSURE() = AiOrenNayarBSDF(sg, Kd, sg->Nf);
```

### Opacity Shader

```
// old
sg->out.RGB = color;
sg->out_opacity = opacity;

// new, opacity must be premultiplied into other closures
AtClosureList closures;
closures.add(AiClosureEmission(sg, opacity * color));
closures.add(AiClosureTransparent(sg, 1 - opacity));
sg->out.CLOSURE() = closures;
```

### Matte Shader

```
// old
sg->out.RGBA = AiRGBACreate(color.r, color.g, color.b, alpha);
sg->out_opacity = opacity;

// new, matte alpha is specified through its own closure
AtClosureList closures;
closures.add(AiClosureEmission(sg, opacity * color));
closures.add(AiClosureMatte(sg, opacity * (1 - alpha)));
closures.add(AiClosureTransparent(sg, 1 - opacity));
sg->out.CLOSURE() = closures;
```

### Volume Shader

```
// old
AiShaderGlobalsSetVolumeAttenuation(sg, attenuation);
AiShaderGlobalsSetVolumeScattering(sg, scattering, anisotropy);
AiShaderGlobalsSetVolumeEmission(sg, emission);

// new
sg->out.CLOSURE() = AiClosureVolumeHenyeyGreenstein(sg, attenuation - scattering,
scattering, emission, anisotropy);
```

#### Atmosphere Shader

```
// old
sg->Vo = emission;
sg->out.RGB = transparent * sg->Ci + emission;
sg->out.RGBA.a = matte_alpha;

// new
AtRGB matte = (1 - transparent) * (1 - matte_alpha);
sg->out.CLOSURE() = AiClosureVolumeAtmosphere(emission, transparent, matte);
```

## Light Filters

Light filters are arbitrary shaders that can modify, or filter, the incoming illumination from a given light. Light filters are just regular shaders attached to a light's "filters" parameter. Compared to an old-style RenderMan light shader, the only thing that an Arnold light filter cannot do is modify the light emitting geometry and its associated sampling.

Arnold currently ships with four built-in light filter shaders, which are explained in detail in the [MtoA docs](#). Here is an example using the built-in gobo shader:

```
image
{
    name myimage
    filename polkadots.jpg
}

gobo
{
    name mygobo
    slidemap mytexture
    rotate 30
    scale_s 2
    scale_t 2
}

spot_light
{
    name myspot
    ...
    filters mygobo
    # you can also attach a stack of filters:
    # filters 3 1 NODE gobol gobo2 gobo3
}
```

But you can also write your own light filter. You can use some shader globals inputs (such as the light direction `sg->light_filter->Ld`, the distance `sg->light_filter->Ldist`, or, for a spot light, `sg->u/v`) and overwrite the unoccluded light intensity `sg->light_filter->Liu` in-place. Below is a simple example that modulates the light intensity with a Perlin noise procedural texture:

```

#include <ai.h>
#include <string.h>

AI_SHADER_NODE_EXPORT_METHODS(SimpleLightFilterMethods);

node_parameters { }
node_initialize { }
node_update { }
node_finish { }

shader_evaluate
{
    // test if we are running as a light filter
    if (sg->light_filter)
    {
        AtVector2 p(sg->u, sg->v);
        sg->light_filter->Liu *= (AiPerlin2(p * 50) + 0.5f);
    }
}

node_loader
{
    if (i > 0)
        return false;
    node->methods      = SimpleLightFilterMethods;
    node->output_type  = AI_TYPE_RGB;
    node->name         = "simple_light_filter";
    node->node_type    = AI_NODE_SHADER;
    strcpy(node->version, AI_VERSION);
    return true;
}

```

### simple light filter.ass

[Expand source](#)

```

options
{
    AA_samples 5
    GI_diffuse_depth 2
    GI_diffuse_samples 3
}

persp_camera
{
    name mycamera
    position 5 3.0 5
    look_at 0 0.8 1
    up 0 1 0
    fov 30
}

skydome_light
{
    name mysky
}

```

```
color 0.8 0.9 1.0
intensity 0.5
}

lambert
{
name mylambert
Kd 0.5
}

plane
{
name myplane
normal 0 1 0
shader mylambert
}

sphere
{
name myspherel
center 0 1 0
radius 1
shader mylambert
}

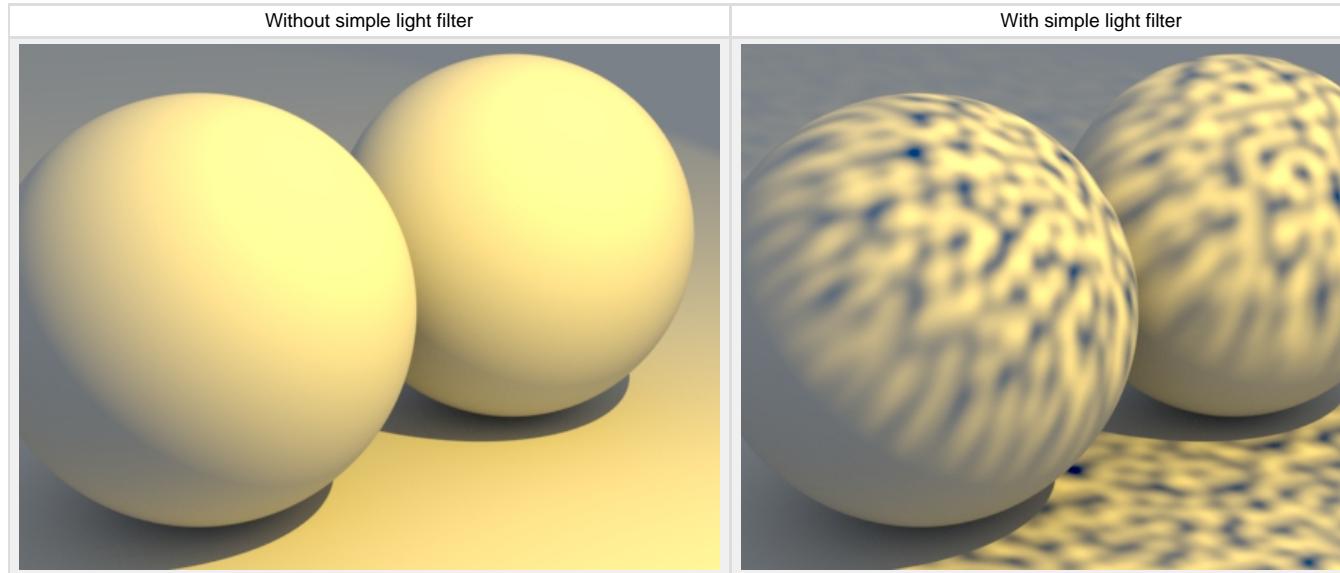
sphere
{
name mysphere2
center 0 1 2
radius 1
shader mylambert
}

spot_light
{
name mylight
look_at 0 1 0
position 4 5 1
intensity 3.141
color 1.0 0.7 0.1
cone_angle 65
penumbra_angle 2
exposure 6
filters my_light_filter
}

simple_light_filter
```

```
{  
    name my_light_filter  
}
```

The above example produces the following image:



## Light Loops

Typically, it is best to avoid doing any manual light integration and leave it up to Arnold, to take advantage of an optimized implementation that can continue improving without changes to shaders and BSDFs, and to support light path expressions. However, a lower level direct light sampling API is also provided.

The [BSDF API](#) can be used to implement a wide range of BSDFs, which can then be output as a [closure](#) or integrated using [AiBSDFIntegrate](#).

```
#include <ai.h>

AI_SHADER_NODE_EXPORT_METHODS(DiffuseMtd)

enum DiffuseParams {
    p_color,
};

node_parameters
{
    AiParameterRGB("color", 0.8f, 0.8f, 0.8f);
}

node_initialize
{
    // create 3D sampler for BSDF sampling
    const int seed = 3158863998;
    const int nsamples = 4;
    const int ndim = 3;
    AtSampler *sampler = AiSampler(seed, nsamples, ndim);
    AiNodeSetLocalData(node, sampler);
}

node_update
{
}

node_finish
{
    AtSampler *sampler = (AtSampler*)AiNodeGetLocalData(node);
    AiSamplerDestroy(sampler);
}

static float PowerHeuristic(float pdf_a, float pdf_b)
{
    return AiSqr(pdf_a) / (AiSqr(pdf_a) + AiSqr(pdf_b));
}

static AtRGB DirectLightMIS(AtShaderGlobals *sg, AtSampler *sampler,
                           AtBSDF *bsdf)
{
    const AtRGB weight = AiBSDFGetWeight(bsdf);
    const AtBSDFMethods *methods = AiBSDFGetMethodbsdf();
    AtRGB result = AI_RGB_BLACK;

    // initialize BSDF
    methods->Init(sg, bsdf);
```

```

// prepare light integration
AiLightsPrepare(sg);

// integrate each BSDF lobe separately
const int num_lobes = AiBSDFGetNumLobes(bsdf);
AtBSDFLobeSample *lobe_sample = new AtBSDFLobeSample[num_lobes];

for (int lobe = 0; lobe < num_lobes; lobe++)
{
    const AtBSDFLobeMask lobe_mask = 1 << lobe;
    uint16_t ray_type = AiBSDFGetLobes(bsdf)[lobe].ray_type;
    float nsamples_bsdf = 0;

    // integrate MIS BSDF samples, if there are lights that can benefit from them
    if (AiLightsTraceRayTypes(sg) & ray_type)
    {
        AtSamplerIterator *iterator = AiSamplerIterator(sampler, sg);
        nsamples_bsdf = AiSamplerGetSampleCount(iterator);
        AtVector rnd;

        while (AiSamplerGetSample(iterator, &rnd.x))
        {
            AtVectorDv wi;
            int lobe_index;

            if (methods->Sample(bsdf, rnd, 0.0f, lobe_mask, true, wi, lobe_index,
lobe_sample))
            {
                AtRGB bsdf_weight = weight * lobe_sample[0].weight;
                float bsdf_pdf = lobe_sample[0].pdf * nsamples_bsdf;

                AtLightSample *hits;
                int num_hits = AiLightsTrace(sg, wi.val, ray_type, hits);

                for (int i = 0; i < num_hits; i++)
                {
                    AtLightSample& light_sample = hits[i];

                    float light_influence = AiLightGetInfluence(sg, light_sample.Lp,
ray_type);
                    if (light_influence == 0.0f)
                        continue;

                    float mis_weight = PowerHeuristic(bsdf_pdf, light_sample.pdf);
                    result += (light_influence * mis_weight) * light_sample.Li
                            * bsdf_weight / nsamples_bsdf;
                }
            }
        }
    }

    // integrate MIS light samples
    AtLightSample light_sample;
    while (AiLightsGetSample(sg, light_sample))
    {
        float light_influence = AiLightGetInfluence(sg, light_sample.Lp,
ray_type);

```

```

    if (light_influence == 0.0f)
        continue;

    if (light_sample.trace_ray_types & ray_type)
    {
        // Use MIS
        methods->Eval(bsdf, light_sample.Ld, lobe_mask, true, lobe_sample);
        AtRGB bsdf_weight = weight * lobe_sample[0].weight *
lobesample[0].pdf;
        float bsdf_pdf = lobe_sample[0].pdf * nsamples_bsdf;
        float mis_weight = PowerHeuristic(light_sample.pdf, bsdf_pdf);
        result += (light_influence * mis_weight) * bsdf_weight
                    * light_sample.Li / light_sample.pdf;
    }
    else
    {
        // No MIS
        methods->Eval(bsdf, light_sample.Ld, lobe_mask, false, lobe_sample);
        AtRGB bsdf_weight = weight * lobe_sample[0].weight *
lobesample[0].pdf;
        result += light_influence * bsdf_weight
                    * light_sample.Li / light_sample.pdf;
    }
}
delete[] lobe_sample;
return result;
}

shader_evaluate
{
    // early out for shadow rays and black
    if (sg->Rt & AI_RAY_SHADOW)
        return;

    AtRGB weight = AiShaderEvalParamRGB(p_color);
    if (AiColorIsSmall(weight))
        return;

    // create and integrate BSDF
    AtBSDF *bsdf = AiOrenNayarBSDF(sg, weight, sg->Nf);
    AtSampler *sampler = (AtSampler*)AiNodeGetLocalData(node);
    sg->out.RGB() = DirectLightMIS(sg, sampler, bsdf);
}

node_loader
{
    if (i>0)
        return false;

    node->methods      = DiffuseMtd;
    node->output_type  = AI_TYPE_RGB;
    node->name          = "diffuse";
    node->node_type     = AI_NODE_SHADER;
}

```

```
    strcpy(node->version, AI_VERSION);
    return true;
}
```

## Light Path Expression AOVs

For Arnold 5.0 [shaders using closures](#), light path expressions are used to output light into specific AOVs. Rather than writing to specific AOVs from shaders, LPEs can be used to extract specific light paths using regular expressions.

There is also an [Introduction to Light Path Expressions](#) tutorial.

### Builtin LPEs

The following light path expression AOVs are built-in.

Name	Expression	
RGB	C.*	Beauty AOV with all light paths
direct	C[DSV]L	Direct light from all surfaces and volumes
indirect	C[DSV][DSVOB].*	Indirect light from all surfaces and volumes
emission	C[LO]	Lights and emissive objects directly visible from the camera
background	CB	Background emission
diffuse	C<RD>.*	Diffuse reflection
specular	C<RS['coat']>.*	Specular reflection
coat	C<RS['coat']>.*	Coat reflection
transmission	C<TS>.*	Specular transmission
sss	C<TD>.*	Subsurface scattering and diffuse transmission
volume	CV.*	Volume scattering
albedo	C[DSV]A	Albedo (reflectivity)
diffuse_direct	C<RD>L	Diffuse direct light
diffuse_indirect	C<RD>[DSVOB].*	Diffuse indirect light
diffuse_albedo	C<RD>A	Diffuse albedo
specular_direct	C<RS['coat']>L	Specular direct light
specular_indirect	C<RS['coat']>[DSVOB].*	Specular indirect light
specular_albedo	C<RS['coat']>A	Specular albedo
coat_direct	C<RS['coat']>L	Coat direct light
coat_indirect	C<RS['coat']>[DSVOB].*	Coat indirect light
coat_albedo	C<RS['coat']>A	Coat albedo
transmission_direct	C<TS>L	Transmitted / refracted direct light
transmission_indirect	C<TS>[DSVOB].*	Transmitted / refracted indirect light
transmission_albedo	C<TS>A	Transmission albedo
sss_direct	C<TD>L	SSS and diffuse transmission direct light
sss_indirect	C<TD>[DSVOB].*	SSS and diffuse transmission indirect light
sss_albedo	C<TD>A	SSS and diffuse transmission albedo
volume_direct	CVL	Volume direct light
volume_indirect	CV[DSVOB].*	Volume indirect light
volume_albedo	CVA	Volume albedo

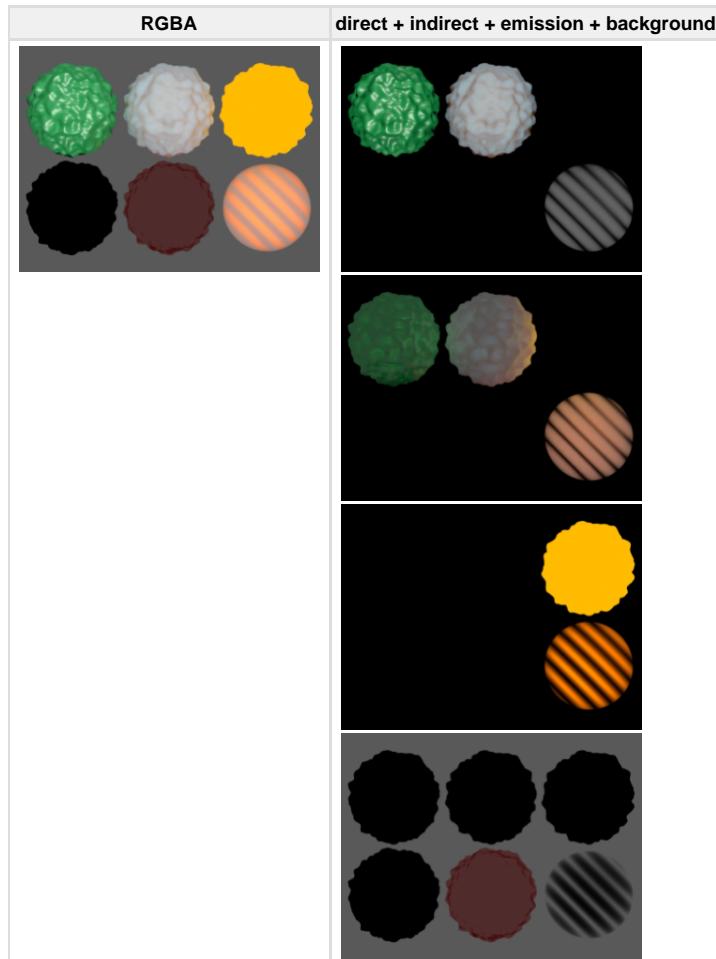
## Composing the Beauty AOV

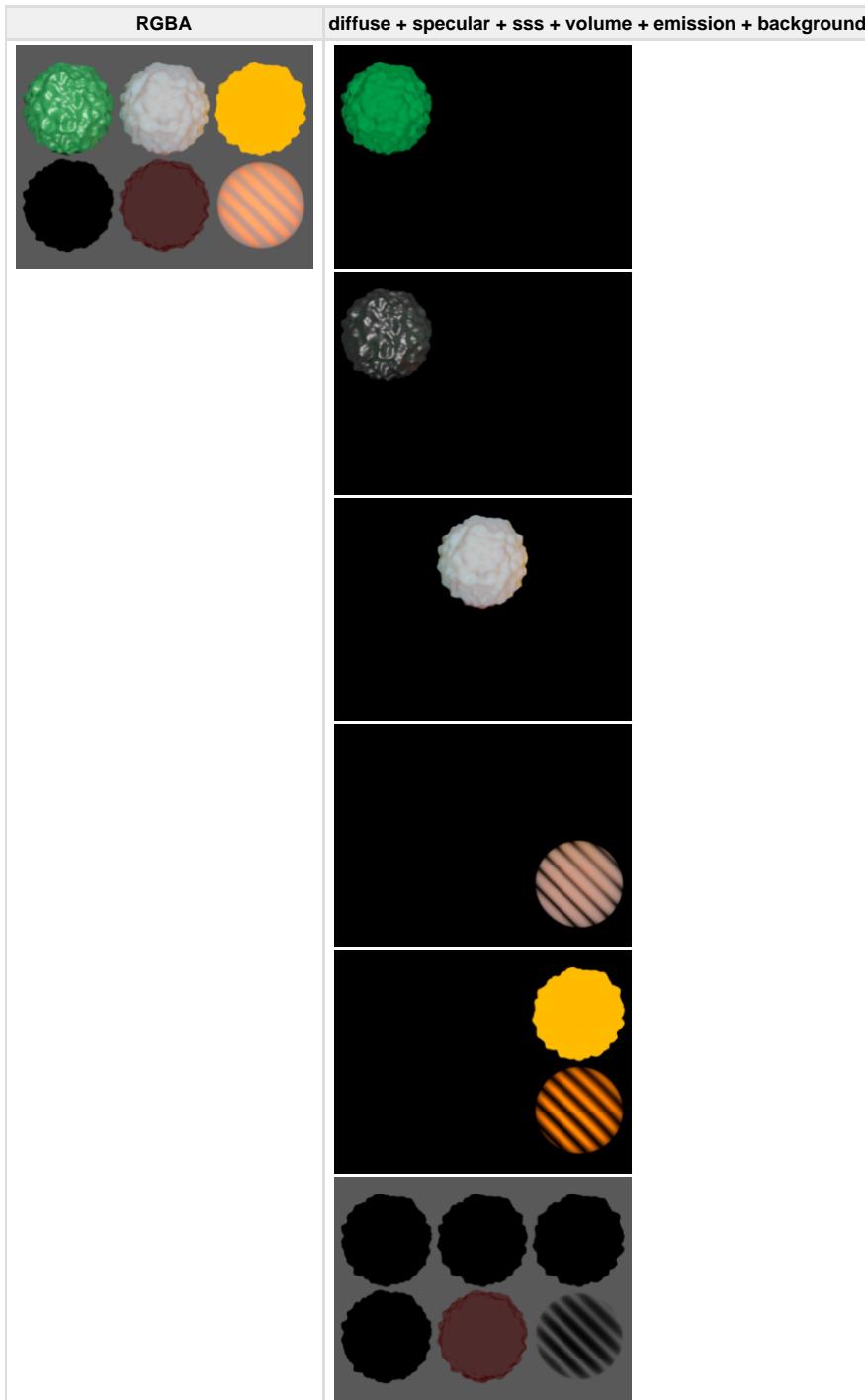
The RGBA beauty AOV can be split into smaller AOVs where each contains part of the lighting. In compositing these AOVs can then be individually modified and added together to get the full beauty AOV.

More AOVs give more control in compositing, but also extra work to handle, and they take up more memory and disk space, especially combined with light groups. Some example sets of additive AOVs for the full beauty AOV are:

- direct, indirect, emission, background
- diffuse, specular, coat, transmission, sss, volume, emission, background
- diffuse\_direct, diffuse\_indirect, specular\_direct, specular\_indirect, coat, transmission, sss, volume, emission, background

Simply adding together such AOVs is all that is needed for the beauty AOV. The albedo AOVs are not needed to reconstruct the beauty AOV, but may be used for example to get just the lighting without the surface texture, by dividing diffuse by diffuse\_albedo, or for denoising just the lighting while keeping the texture detail intact.





## Syntax

All expressions must start with a camera event `c`. After that, there can be zero or more scattering events, until a light, object or background is hit. The syntax is similar to regular expressions. The following events and operators are supported.

## Events

Symbol	
C	Camera
L	Light emission from all lights
<L. 'groupname' >	Light emission from light in AOV groupname
<L. 'default' >	Light emission from lights with no AOV group assigned
O	Surface or volume object emission
B	Background emission
A	Albedo

## Scattering Events

Symbol	
D	Diffuse reflection, transmission or subsurface scattering
<RD>	Diffuse reflection
<TD>	Diffuse transmission or subsurface scattering
S	Specular reflection or refraction
<RS>	Specular reflection
<TS>	Specular refraction
V	Volume scattering

## Operators

Expression	
AB	A followed by B
[ABC]	A, B or C
( (AB)   (CD) )	AB or CD
[^A]	Any event except A
A*	0 or more A events
A+	1 or more A events
.	Any event
'myevent'	Custom event

## Custom Expressions

Custom light path expressions are added to `options.light_path_expressions`, and can then be used as AOVs in `options.outputs`.

Custom expressions can override built-in expressions with the same name, if a different behavior is desired.

```
light_path_expressions 2 1 STRING
"caustics CDS.*"
"diffuse_albedo CDA"
outputs 3 1 STRING
"RGBA RGBA filter driver"
"caustics RGB filter driver"
"diffuse_albedo RGB filter driver"
```

## Light Groups

Besides the `<L. 'groupname' >` syntax, any built-in or custom LPE can be split into multiple AOVs to output a subset of lights with a specific AOV group assigned in the `light.aov` parameter. For this, a postfix must be added to the LPE AOV name in `options.outputs`.

```

outputs "diffuse RGB filter driver"           # all lights in one AOV
outputs "diffuse_* RGB filter driver"         # output multiple AOVs, one for each
light group
outputs "diffuse_groupname RGB filter driver" # only light in AOV group
"groupname"
outputs "diffuse_default RGB filter driver"   # only lights with no AOV group
assigned

```

## Custom Labels

BSDF lobes, BSSRDFs and emission closures may be tagged with a custom label to output them to separate AOVs.

For built-in shaders standard hair is labeled with 'hair', and the standard surface coat is labeled with 'coat'. All other built-in shaders components have no labels. So the following expressions could be used for example:

```

light_path_expressions 2 1 STRING
"hair C'hair'.*"
"specular_except_coat C<RS[^'coat']>.**"

```

Custom shaders could for example label BSSRDFs and emission, and then use expressions like these:

```

light_path_expressions 2 1 STRING
"sss_skin C<TD'skin'>.**"
"emission_fire C<O.'fire'>"

```

## OSL Shaders

OSL is a shading language designed for modern physically based rendering, supported starting from Arnold 5.0. It can be used to write shaders that work in Arnold and other renderers. It is an alternative to C++ shaders, with a higher level API that makes writing shaders simpler while at the same time providing advanced optimizations.

### Getting Started

OSL shaders are written in .osl files, with each file corresponding to a single Arnold shader node. Here is a simple example:

#### gamma.osl

```
shader gamma(
    color Cin = color(0, 0, 0),
    float exponent = 1,
    output color Cout = color(0, 0, 0))
{
    Cout = pow(Cin, 1/exponent);
}
```

The shader name `gamma` must match the `gamma.osl` filename. The shader defines input and output parameters much like Arnold C++ shader nodes, but using a more compact syntax. Here `Cin` and `gamma` are input parameters, while `Cout` is the output parameter. All parameters must be initialized with default values.

### Compilation

Shaders are compiled automatically when they are found by Arnold in the shader searchpath, from .osl files to .oso files in the same directory. This happens if no corresponding .oso file exists, or if the .osl file was modified since the last compilation.

Alternatively, shaders can be manually compiled using the `oslc` program included with Arnold.

```
$ oslc gamma.osl
Compiled gamma.osl -> gamma.oso
```

.oso files contain intermediate code that is operating system and CPU agnostic. At render time, OSL translates shader networks into machine code on the fly, and in the process heavily optimizes shaders and networks with full knowledge of the shader parameters and other runtime values that could not have been known when the shaders were compiled from .osl source code.

### Installation

.osl and .oso files are installed the same way as .so and .dll shader files: place them anywhere in the shader searchpath, and Arnold will find them.

Using `kick` we can verify that Arnold detects the shader and its parameters.

```

$ kick -info gamma
node:      gamma
type:      shader
output:    RGB
parameters: 3
filename:  gamma.oso
version:   4.2.12.0
Type      Name          Default
-----
RGB       Cin           0, 0, 0
FLOAT    exponent      1
STRING   name

```

## Manually Loading Shaders

Besides the automatically registered OSL shaders in the search path, it is possible to load shaders manually through an `osl` node. This can be useful to insert expressions into a shader network, or to get more control over which shaders are loaded and how their parameters should be interpreted.

`osl` accepts a `shadernode` parameter to specify the shader, without the `.osl` or `.oso` file extension. As soon as the `shadernode` parameter is set, the OSL shader parameters are added to the node, with a `param_` prefix.

```

osl
{
  name myshader
  shadernode somefolder/test
  param_value 0.5
}

```

For inserting expressions into the network, the `.OSL` or `.OSO` code can be embedded as well:

```

osl
{
  name myshader
  code "shader test(float value = 0, output float result = 0)
  {
    result = value * 10;
  }"
  param_value 0.5
}

```

## Supported Features

### Parameters

OSL shader parameters are converted to the corresponding Arnold parameters where possible. Arnold only supports one output parameter per node, so if there are multiple output parameters the first parameter will be used.

OSL Type	Arnold Type
int	INT
int (with metadata)	BOOLEAN
int (with metadata)	ENUM
float	FLOAT
color	RGB
color	RGBA
point	VECTOR
vector	VECTOR
normal	VECTOR
point	POINT2
matrix	MATRIX
array of any type	ARRAY
closure color	CLOSURE
struct	POINTER

Boolean and enum parameter types are created with OSL metadata on integers.

```
#define OPTION_A 0
#define OPTION_B 1
#define OPTION_C 2

shader example(
    int booleanvalue = 0 [[ string widget = "boolean" ]],
    int enumvalue = 0 [[ string widget = "popup", string options =
"OptionA|OptionB|OptionC" ]])
{
    if (booleanvalue)
        ...
    if (enumvalue == OPTION_B)
        ...
}
```

## Attributes

Node parameters and user data are available through `getattribute()`.

```
// object parameter
int id;
getattribute("id", id);

// object user data
color Cs;
getattribute("Cs", Cs);

// parameter of another node
int AA_samples;
getattribute("options", "AA_samples", AA_samples);
```

### Standard attributes

Using the following attributes, if the `getattribute()` function specifies an `objectname` parameter, the value specific to that object is retrieved. If

no specific object is named, the current object is implied.

Name	Type	Description
"geom:type"	string	Object type
"geom:name"	string	Object name
"geom:bounds"	point[2]	Object bounding box in world space (min, max)
"geom:objbounds"	point[2]	Object bounding box in object space (min, max)

### Standard camera attributes

Using the following attributes, if the `getattribute()` function specifies an `objectname` parameter and it is the name of a valid camera, the value specific to that camera is retrieved. If no specific camera is named, the global or default camera is implied.

Name	Type	Description
"camera:screen_window"	int[2]	Image resolution
"camera:pixelaspect"	float	Pixel aspect ratio
"camera:projection"	string	Camera type
"camera:fov"	float	Field of view
"camera:clip_near"	float	Near clip distance
"camera:clip_far"	float	Far clip distance
"camera:clip"	float[2]	Near and far clip distances
"camera:shutter_open"	float	Shutter open time
"camera:shutter_close"	float	Shutter close time
"camera:shutter"	float[2]	Shutter open and close times
"camera:screen_window"	float[4]	Screen window (xmin, ymin, xmax, ymax)

### Shader Globals

Arnold supports most OSL shader globals, like `P`, `u`, `v`, `N`, `Ng` and `time`. Their meaning is the same as in the C++ shading API.

`Ps` (for light filters), `surfacearea()`, `dtime` and `dPdt` shader globals are not supported currently.

All shader globals are considered to be read-only, we do not support writing to `Ci` and `P` to output closures or displacement. Output parameters should be used instead.

### Textures

Textures are accessed through the built-in `texture()` and `gettextureinfo()` functions. The `texture()` function accepts an optional `colorspace` argument to indicate the texture's color space to convert from.

Note that unlike the Arnold C++ shading API, the texture origin is assumed to be in the top left corner rather than the bottom left corner, for consistency with the OSL standard. To match, the `v` coordinate can be flipped to `1 - v`, or `floor(v) + 1 - mod(v, 1)` in case of UDIM textures.

```
// look up texture color
color tex = texture(filename, u, v);

// query texture resolution
int resolution;
gettextureinfo(filename, "resolution", resolution);
```

### Volume Channels

Volume channels are available through `texture3d()` using object space coordinates.

```
point Po = transform("object", P);
float density = texture3d("density", Po, "interp", "bicubic");
```

## Closures

The following [closures](#) are supported, matching the closures in the C++ shader API.

### Supported Closures

```
closure color emission();
closure color background();
closure color diffuse(normal N);
closure color oren_nayar (normal N, float sigma);
closure color translucent(normal N);
closure color microfacet(string distribution, normal N, vector U,
                         float xalpha, float yalpha, float eta, int refract);
closure color microfacet(string distribution, normal N,
                         float alpha, float eta, int refr);
closure color reflection(normal N, float eta);
closure color reflection(normal N);
closure color refraction(normal N, float eta);
closure color transparent();
closure color holdout();
closure color empirical_bssrdf(vector mfp, color albedo);
closure color volume_absorption();
closure color volume_emission();
closure color volume_henyey_greenstein(color absorption, color scattering,
                                         color emission, float g);
```

Example shader outputting a closure:

```
shader simple_diffuse(
    color albedo = 0.8,
    float opacity = 1.0,
    output closure color result = 0)
{
    result = opacity * albedo * diffuse(N) + (1 - opacity) * transparent();
```

## Trace

The OSL `trace()` function is supported, for tracing probe rays. The `shade` argument to perform shading is not supported. The `traceset` argument is supported, using an inclusive traceset by default, and exclusive if the traceset name is prefixed with a `-` character.

Information about the hit may be retrieved with `getmessage()`, supporting `hit`, `hitdist`, `P`, `N`, `u`, `v`, and arbitrary user data and parameter lookups on the object that was hit.

```

int hit = trace(origin, direction, "traceset", tracesetname);

if (hit)
{
    // query hit distance
    float hitdist;
    getmessage("trace", "hitdist", hitdist);

    // query parameter on object that was hit
    string name;
    getmessage("trace", "name", name);

    // query user data on position where object was hit
    color Cs;
    if (getmessage("trace", "Cs", Cs))
        ...
}

```

## Raytype

The OSL `raytype(<ray type>)` function returns 1 if the ray being shaded is of the given type, or 0 if the ray is not of that type or if the ray type name is not recognized by the renderer.

Supported ray types
'camera"
"shadow"
'diffuse_transmit"
'specular_transmit"
'volume_scatter"
'diffuse_reflect"
'specular_reflect"
'subsurface"

## Performance

OSL and C++ shaders can be linked into a single shader network. However there is a small overhead (perhaps 1-2 % overall render time) in connecting the output of an OSL shader into a C++ shader. For the OSL optimizer to be able to do aggressive whole network optimizations, as many OSL shader nodes as possible should be used.

Runtime compilation and optimization of OSL shaders happens the first time the shader is evaluated, during rendering. This increases the time to first pixel, but can pay off in reduced render time overall.

## Debugging

The OSL\_OPTIONS environment variable can be used to debug common errors in shaders or print more detailed information:

```
# Add (expensive) code to find array out of bounds access, NaN/Infs and use of
# uninitialized variables
OSL_OPTIONS="range_checking=1,debug_nan=1,debug_uninit=1"

# Issue info messages for every shader compiled
OSL_OPTIONS="compile_report=1"

# Number of warning calls that should be processed per thread
OSL_OPTIONS="max_warnings_per_thread=100"
```

## More Information

- [Open Shading Language introduction](#)
- [Open Shading Language specification](#)

# Shader Data Type Conversions

When shader parameters of different types are connected together, some allowance is made for parameters to be connected when they have different types. Here are the rules for automatic parameter conversions.

Key:

(empty)	conversion is disallowed/ignored
copy	copy (no need to convert)
expand	all components set to the same source value
shallow copy	for pointer types (pointer, node, array, matrix), copy the pointer, not the actual value
boolean cast	zero means false, non-zero means true, and vice-versa
upcast	smaller integer type just gets stuffed into a larger integer type
downcast	larger integer type gets truncated into a smaller integer type
average	the average of the components are taken

				<u>TARGET</u>	
<u>SOURCE</u>	string	node	pointer	array	matrix
enum					
boolean					
byte					
int					
uint					
float					
RGB					
RGBA					
vector					
point					
point2					
string	copy				
node		shallow copy			
pointer		shallow copy			
array			shallow copy		
matrix				shallow copy	

A few notes:

- The node type (AI\_TYPE\_NODE) is currently resolved at scene creation time, and node-to-node shader connections are not formally allowed since they internally are turned into a connection based on the output type of the shader node. However, whenever one wants to pass a node along, they can pass it as a generic pointer type instead using other mechanisms such as shader message passing. The vast majority of the time, if you are using pointer passing inside of your shaders, however, you should re-think your design.
- No automatic conversions exist between integer types and floating point types.
- No automatic conversions exist between enumerations and any other type.

## Trace Sets

Trace sets give control over which rays are affected by which object. There are two types of trace sets:

- When a trace set is *exclusive*, rays are traced against all geometry except the tagged nodes.
- When a trace set is *inclusive*, rays are traced against tagged nodes, but also against nodes that are not tagged at all (see example .ass below).

Here is an example shader and .ass file using inclusive tracesets for a probe ray and a diffuse BSDF.

### traceset.cpp

```
#include <ai.h>

AI_SHADER_NODE_EXPORT_METHODS(TracesetMethods);

node_parameters { }
node_initialize { }
node_update { }
node_finish { }

shader_evaluate
{
    const AtString probe_set("probe_set");
    const bool probe_set_inclusive = true;

    const AtString diffuse_set("diffuse_set");
    const bool diffuse_set_inclusive = true;

    AtRGB diffuse_color(0.5f, 0.5f, 0.5f);

    // trace probe ray along normal, turning the surface red on hits
    AiShaderGlobalsSetTraceSet(sg, probe_set, probe_set_inclusive);
    AtRay ray = AiMakeRay(AI_RAY_SHADOW, sg->P, &sg->Nf, AI_BIG, sg);
    AtScrSample sample;
    if (AiTrace(ray, AI_RGB_WHITE, sample))
        diffuse_color = AI_RGB_RED;
    AiShaderGlobalsUnsetTraceSet(sg);

    // diffuse BSDF
    AiShaderGlobalsSetTraceSet(sg, diffuse_set, diffuse_set_inclusive);
    sg->out.CLOSURE() = AiOrenNayarBSDF(sg, diffuse_color, sg->Nf);
    AiShaderGlobalsUnsetTraceSet(sg);
}

node_loader
{
    if (i > 0)
        return false;

    node->methods      = TracesetMethods;
    node->output_type  = AI_TYPE_CLOSURE;
    node->name         = "traceset";
    node->node_type    = AI_NODE_SHADER;
    strcpy(node->version, AI_VERSION);
    return true;
}
```

### traceset.ass

[Expand source](#)

```
options
{
    xres 1024
```

```

yres 512
AA_samples 6
GI_diffuse_depth 4
GI_specular_depth 4
}

persp_camera
{
    name mycamera
    position 0 1 4
    look_at 0 -0.2 0
    up 0 1 0
}

skydome_light
{
    name myskydome
    intensity 1
    color 1 1 1
    camera 0.0
}

standard_surface
{
    name mystd
    base_color 0.4 0.8 0.4
}

sphere
{
    shader mystd
    matrix
        1 0 0 0
        0 1 0 0
        0 0 1 0
        -1.5 0 0 0
    trace_sets 1 1 STRING "probe_set"
}

sphere
{
    shader mystd
    matrix
        1 0 0 0
        0 1 0 0
        0 0 1 0
        0 0 0 0
    trace_sets 1 1 STRING "diffuse_set"
}

sphere
{
    shader mystd
    matrix
        1 0 0 0
        0 1 0 0
        0 0 1 0

```

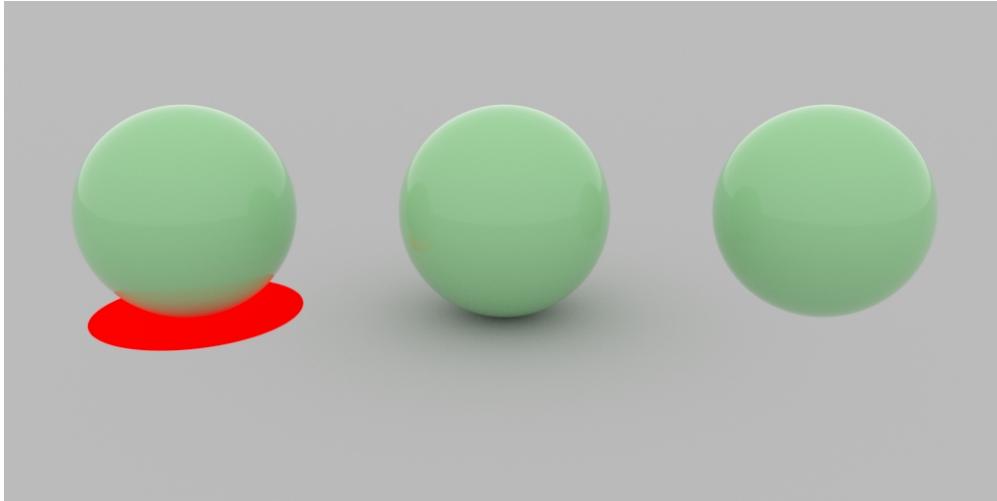
```
    1.5 0 0 0
trace_sets 1 1 STRING ""
}
```

```
traceset
{
    name myshader
}
```

```
plane
{
    name myplane
    normal 0 1 0
```

```
point 0 -0.5 0
shader myshader
{}
```

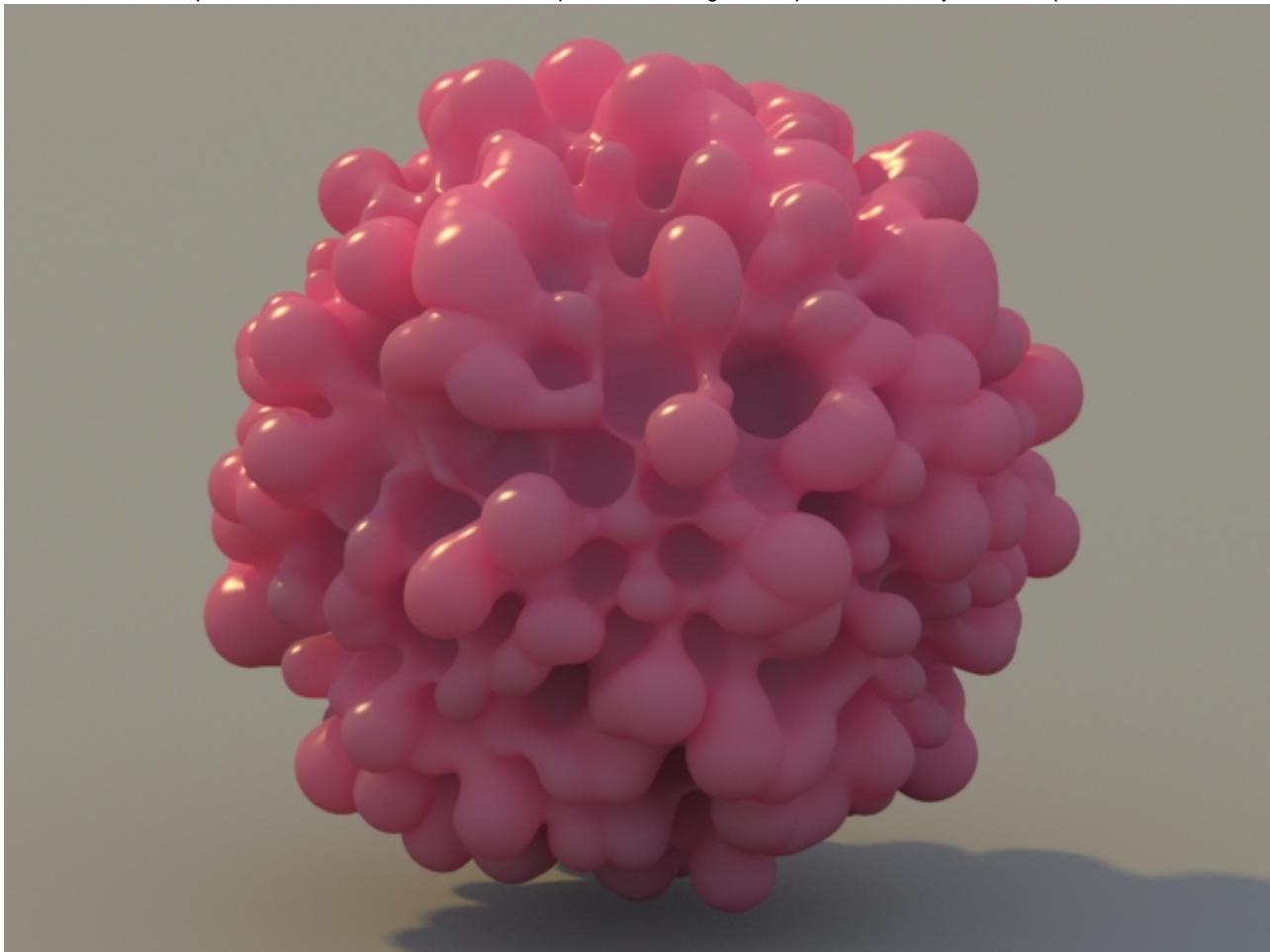
The resulting render looks like this, with the traceset shader assigned to the floor. The left sphere has the probe trace set assigned, which results in a red disk on the floor where the probe rays hit the sphere. The middle sphere has the diffuse trace set assigned, and as a result cast shadows and reflections on the floor. The right sphere has an empty list of tracesets, and affects neither the probe rays nor the diffuse BSDF.



## Vector Displacement

Arnold allows shaders to displace vertices in a polymesh node. The vertices are displaced in the direction and magnitude of the vector returned by the shader.

In this example shader, we compute a nonlinear displacement based on a noise function. This can displace along the normal of a surface like a traditional noise displacement shader, but it also has the option of blooming out the peaks and valleys of the displacement.



This is done by computing a noise function and its delta in the direction of two surface derivatives. This delta determines the amount the derivatives would be deflected by the displacement. By repeatedly deflecting the derivatives, we create a non-linear displacement that arcs out from the surface. This is equivalent to displacing the surface, computing the normal, and displacing again repeatedly.

Here is an animation showing the effect: [vimeo](#) / [download](#).

### Source Code

```
/*
 * nonlinear noise displacement shader
 */
#include <ai.h>
#include <string.h>

enum
{
    p_octaves,
    p_freq,
```

```

    p_amplitude,
    p_bloom,
    p_type
};

AI_SHADER_NODE_EXPORT_METHODS(NonlinNzMethods);

#define ENUM_SCALAR_TYPES { "perlin", "abs_perlin", "recursive", "abs_recursive",
NULL };
#define PERLIN      0
#define ABS_PERLIN   1
#define recursive   2
#define ABS_RECURSIVE 3
const char *types_enum[ ] = ENUM_SCALAR_TYPES;

node_parameters
{
    AiParameterInt ("octaves" , 3);
    AiParameterFlt ("freq"     , 1);
    AiParameterFlt ("amplitude", 1);
    AiParameterFlt ("bloom"    , 1);
    AiParameterEnum("type"     , PERLIN, types_enum);
}

float scalarfunc(AtVector P, int type, int octaves)
{
    float doubler = 1;
    float NzAccum = 0;

    switch (type)
    {
        case PERLIN:
            return AiPerlin3(P);
        case ABS_PERLIN:
            return fabs(AiPerlin3(P));
        case recursive:
            for (int i = 0; i < octaves; i++) {
                NzAccum += AiPerlin3(P*doubler) / doubler;
                doubler *= 2;
            }
            return NzAccum;
        case ABS_RECURSIVE:
            for (int i = 0; i < octaves; i++) {
                NzAccum += fabs(AiPerlin3(P*doubler)) / doubler;
                doubler *= 2;
            }
            return NzAccum;
    }

    return AiPerlin3(P);
}

shader_evaluate
{
    AtVector Ploc, Uloc, Vloc; // noise sample location, and over in U and V
    locations
    float Np, Nu, Nv;          // noise at P, noise at location over in U and V
}

```

```

float Udelt, Vdelt;           // delta in the noise over in U and V
float delta = .01;            // distance delta for noise samples

int octaves = AiShaderEvalParamInt(p_octaves);
float freq = AiShaderEvalParamFlt(p_freq);
float amplitude = AiShaderEvalParamFlt(p_amplitude);
float bloom = AiShaderEvalParamFlt(p_bloom);
int type = AiShaderEvalParamInt(p_type);

AtVector U, V;
if (sg->dPdu != AI_V3_ZERO && sg->dPdv != AI_V3_ZERO)
{
    // tangents available, use them
    U = sg->dPdu;
    V = sg->dPdv;
}
else
{
    // no tangents given, compute a pair
    AiV3BuildLocalFramePolar(U, V, sg->N);
}

if (type > ABS_PERLIN)
{
    // adjust delta to highest frequency in recursive noise
    delta *= pow(.5, octaves) * 2;
}
Ploc = sg->Po * freq;
Uloc = Ploc + U * delta;
Vloc = Ploc + V * delta;

// noise sampled at P, and over in U and V
Np = scalarfunc(Ploc, type, octaves);
Nu = scalarfunc(Uloc, type, octaves);
Nv = scalarfunc(Vloc, type, octaves);

Udelt = (Nu - Np) * bloom;
Vdelt = (Nv - Np) * bloom;

AtVector Pstepped = sg->P;
int steps = 10;
float stepscale = amplitude / steps;

for (int i = 0; i < steps; i++)
{
    // stepdir is the cross product of the derivatives
    AtVector stepdir = AiV3Cross(U,V);
    // deflect the derivatives
    U = AiV3Normalize(U + (stepdir * Udelt * stepscale));
    V = AiV3Normalize(V + (stepdir * Vdelt * stepscale));
    Pstepped += stepdir * Np * stepscale;
}

sg->out.VEC() = Pstepped - sg->P;
}

node_initialize

```

```
{  
}  
  
node_update  
{  
}  
  
node_finish  
{  
}  
  
node_loader  
{  
    if (i > 0) return false;  
  
    node->methods      = NonlinNzMethods;  
    node->output_type  = AI_TYPE_VECTOR;  
    node->name          = "nonlinear_noise";  
    node->node_type     = AI_NODE_SHADER;
```

```

    strcpy(node->version, AI_VERSION);
    return true;
}

```

And here is the .ass file that uses this shader:

```

options
{
AA_samples 3
xres 640
yres 480
GI_diffuse_depth 1
GI_diffuse_samples 3
}

plane
{
name myplane
point 0 -8 0
normal 0 1 0
shader groundshader
}

polymesh
{
name mysph
nsides 6 1 UINT 4 4 4 4 4 4
vidxs 24 1 UINT
0 4 5 1 1 5 6 2 2 6 7 3 3 7 4 0 3 0 1 2 4 7 6 5
vlist 8 1 b64VECTOR

AAB6wwAAAAAAHrDAAB6QwAAAAAAHrDAAB6QwAAAAAAHpDAAB6wwAAAAAAHpDAAB6wwAA+kMAAHrDAAB
6QwAA+kMAAHrDAAB6QwAA+kMAAHpDAAB6wwAA+kMAAHpD
smoothing on
subdiv_type catclark
subdiv_iterations 7
disp_map sphere_disp
disp_padding 50
matrix
0.94693 0 0.321439 0
0 1 0 0
-0.321439 0 0.94693 0
0 0 0 1
shader sphere_surf
}

standard_surface
{
name sphere_surf
base 0.3
base_color 0.8 0.8 1
specular 1
specular_color 0.8 0.8 1
}
```

```
specular_roughness 0.3
subsurface 0.5
subsurface_color 1 0.05 0.2
subsurface_radius 80 80 80
}

nonlinear_noise
{
    name sphere_disp
    type perlin
    freq 0.025
    amplitude 80
    bloom 1
}

standard_surface
{
    name groundshader
    base 1
    base_color 0.4 0.4 0.4
    specular 0
}

persp_camera
{
    name mycamera
    fov 11
    position 3677.0129 1039.1904 597.0592
    look_at 0 250 0
    up 0 1 0
}

point_light
{
    name key
    position -6000 10000 6000
    radius 400
    color 1 0.7 0.2
    intensity 1
    exposure 28
}

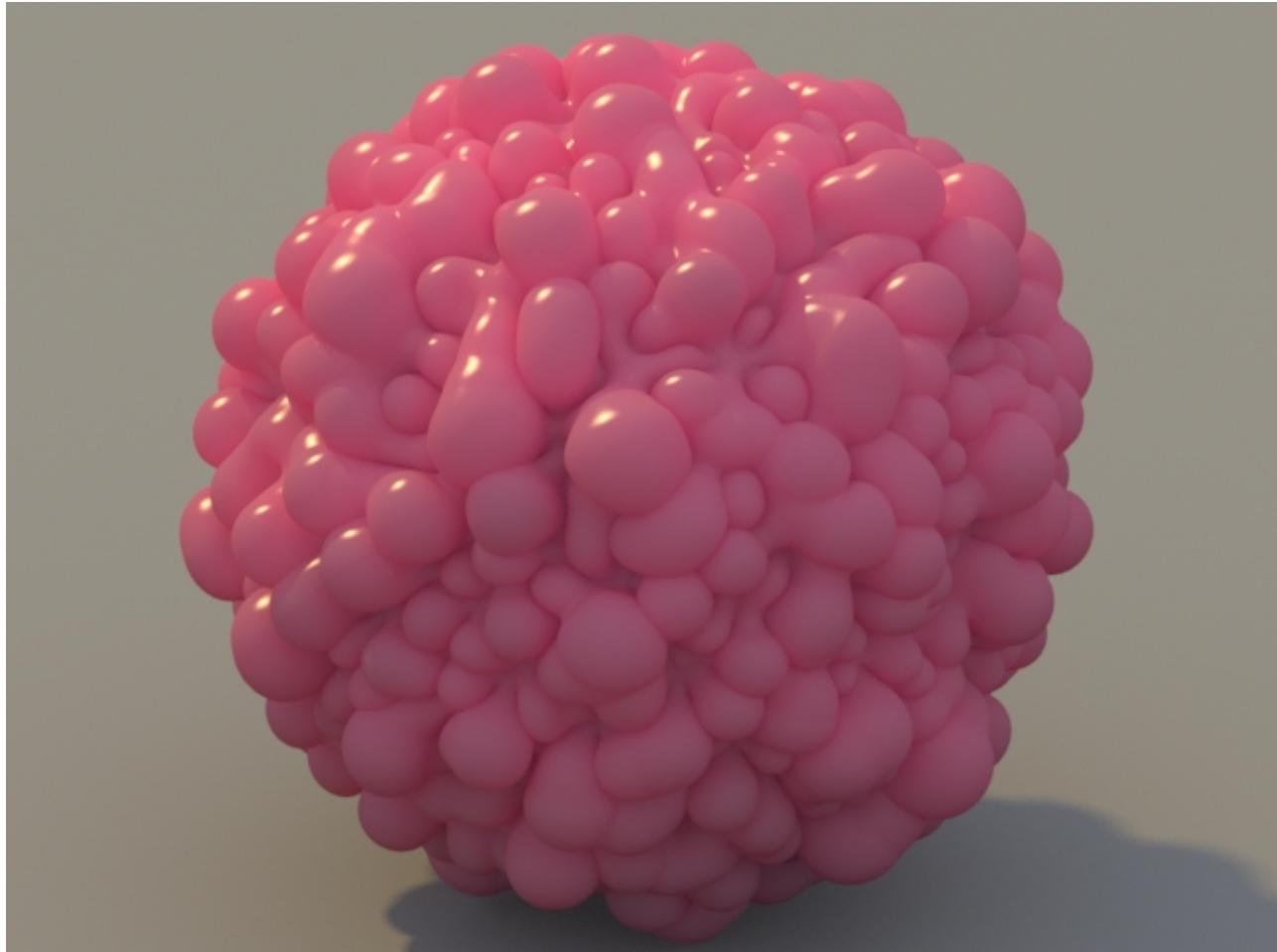
skydome_light
{
    name mysky
```

```
    color 0.7 0.8 0.9  
    intensity 0.9  
}
```

Some examples of looks that can be obtained with this shader, along with the shader settings from the .ass file:

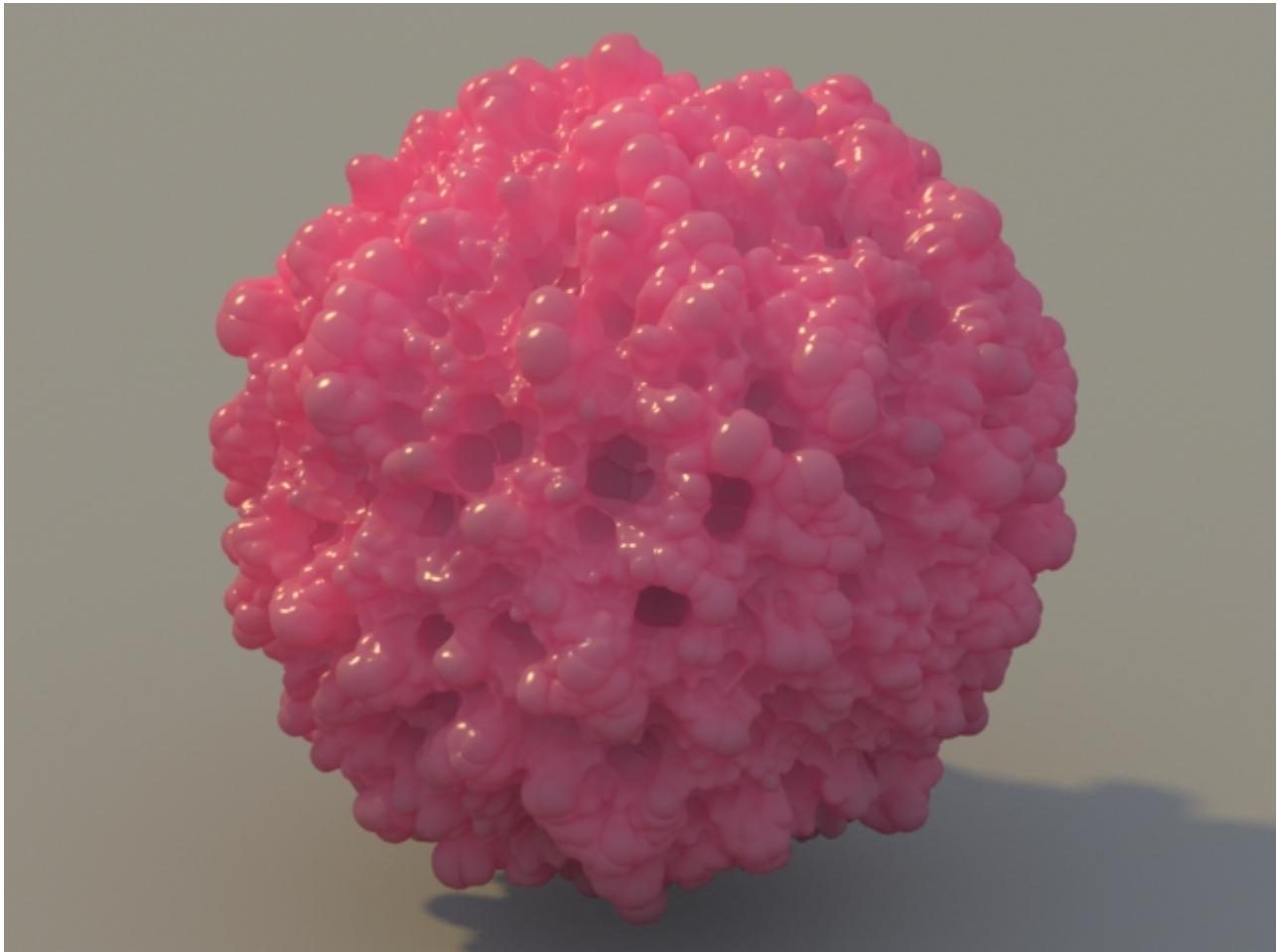
#### **Lumpy Nodules**

```
type abs_perlin  
freq 0.025  
amplitude 80
```



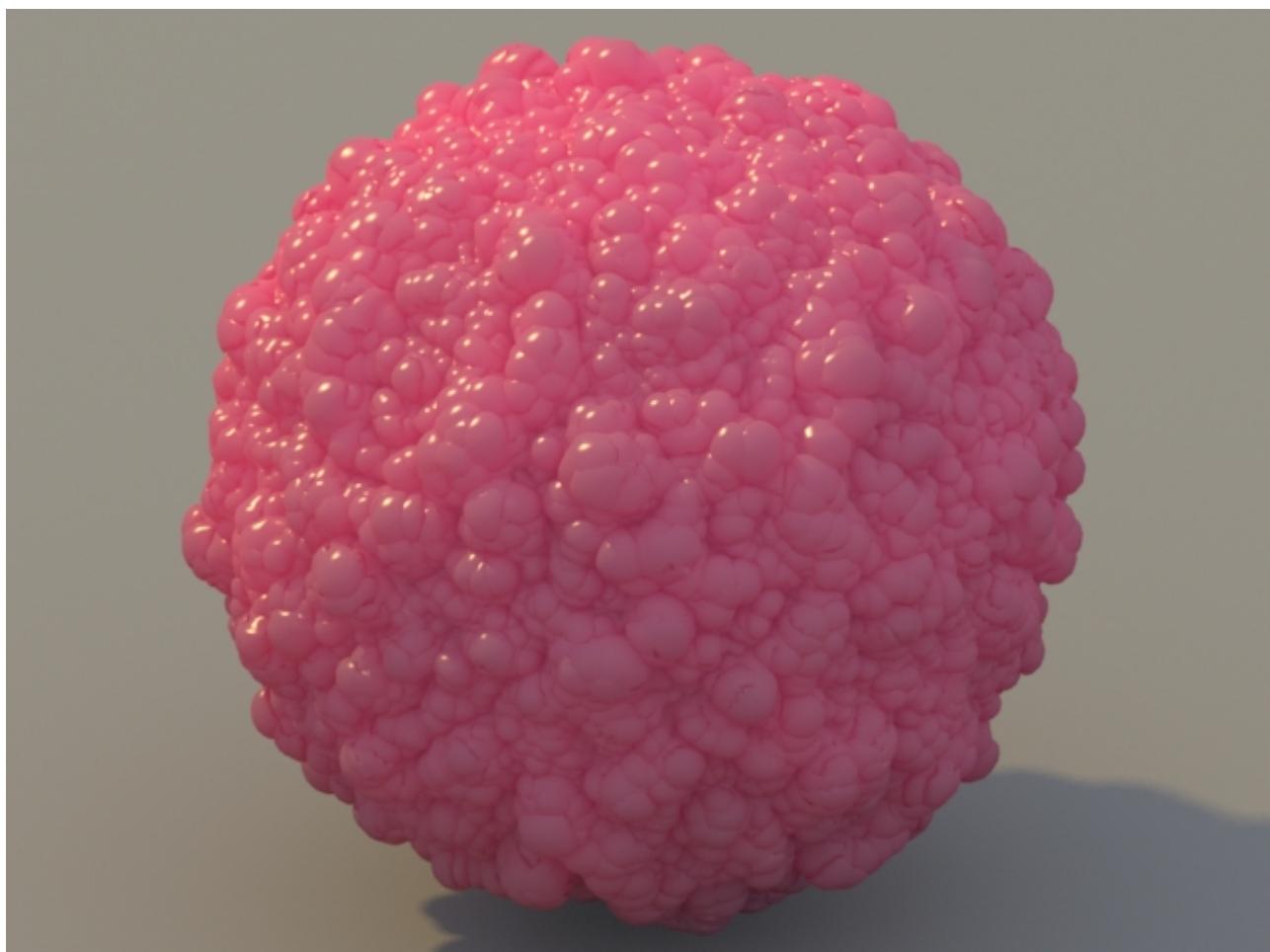
#### **Pomegranate**

```
type recursive
freq 0.025
amplitude 40
bloom 3
```



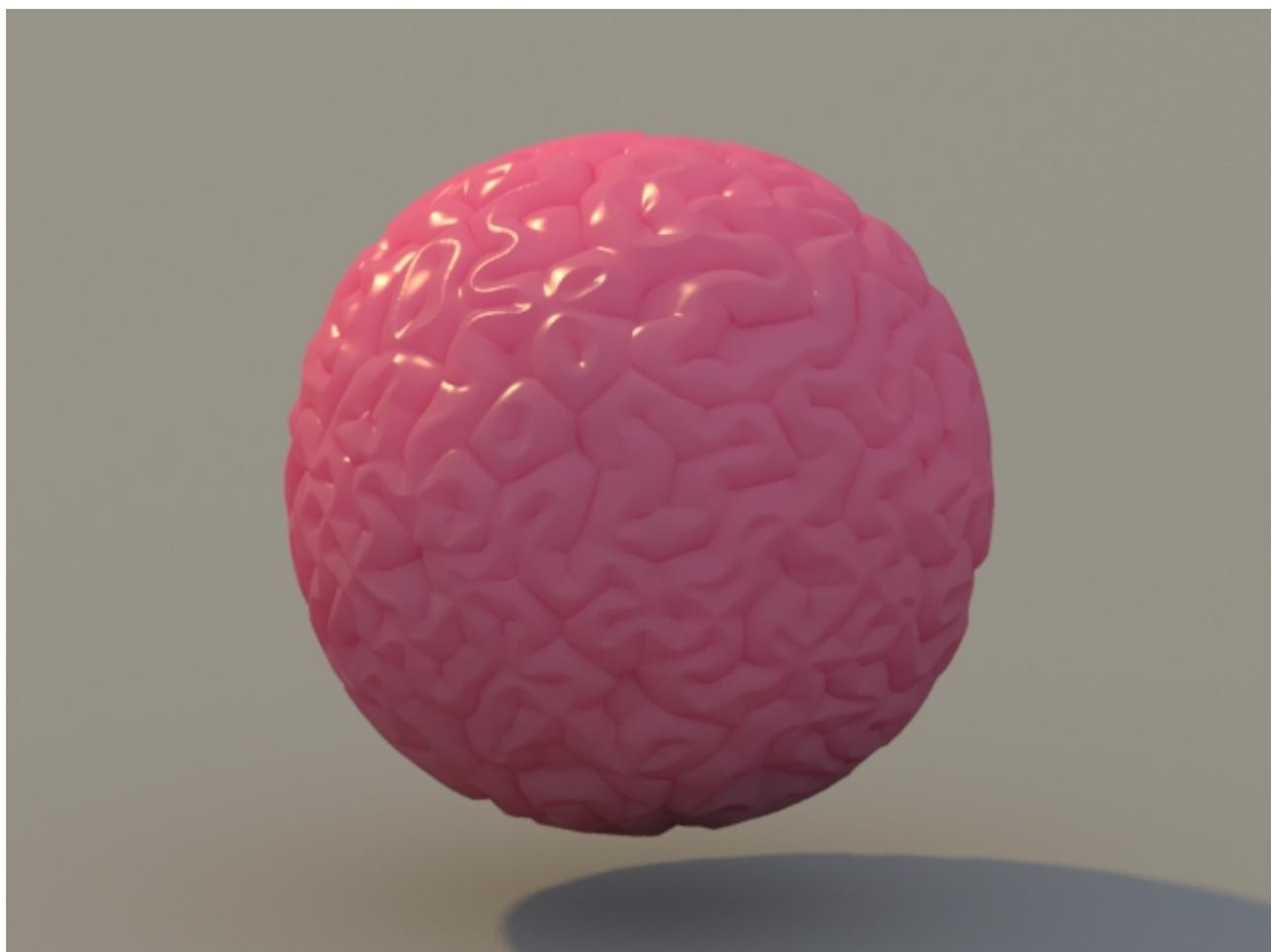
### **Cauliflower**

```
type abs_recursive
freq 0.025
amplitude 40
bloom 3
```



### ***Brain Coral***

```
type abs_perlin
freq 0.025
amplitude -40
bloom -4
```



# Volume Shaders

## Volume Shading API

Volume shaders output closures describing absorption, scattering and emission. These are the available closures:

### Volume Closures

```
AI_API AtClosure AiClosureVolumeAbsorption(const AtShaderGlobals* sg, const AtRGB&
weight);
AI_API AtClosure AiClosureVolumeEmission(const AtShaderGlobals* sg, const AtRGB&
weight);
AI_API AtClosure AiClosureVolumeHenyeyGreenstein(const AtShaderGlobals* sg,
    const AtRGB& absorption, const AtRGB& scattering, const AtRGB& emission, float g
= 0.f);
AI_API AtClosure AiClosureVolumeMatte(const AtShaderGlobals* sg, const AtRGB&
weight);
```

You can think of volumes as a sort of point cloud with infinitesimally small motes. When rays of light traverse a volume they may either hit a mote and be reflected/scattered (think white motes), they may hit a mote and be absorbed (think black motes), or they may traverse the volume without hitting anything at all. Depending on how tightly packed together the motes are, there will be a greater or lesser chance of actually hitting a mote, and this chance increases the greater the distance that the ray traverses in the volume. The weights of the volume closures are the absorption, scattering and emission coefficients, which are rates with unit  $\text{m}^{-1}$ .

The emission coefficient is the rate at which a volume emits light at a given point. A ray traversing a volume with a constant emission coefficient will have radiance added at a rate of **emission\_coefficient \* distance\_traveled**. The light emitted by a volume is visible to global illumination. It will also be affected by any attenuation, out-scattering or absorption effects in the volume.

The scattering coefficient is the rate at which light is scattered (or reflected) at a given point. The greater the rate of scattering, the shorter the average distance a ray of light will travel through a volume before being bounced off of its course. There is an optional parameter for the Henyey-Greenstein closure to describe the mean cosine of the direction of the scattered ray with respect to that of the original ray ( $g$ ). Valid values for  $g$  are anywhere between -1 (full back-scatter) and 1 (full forward-scatter), and by default the mean cosine is set to 0 (isotropic scattering).

The absorption coefficient is the rate at which light is absorbed at a given point. Summing the absorption and scattering coefficients gives the **attenuation coefficient** (also called extinction coefficient), which corresponds to the overall density of the volume.

Volume shaders often do not expose absorption and scattering coefficients as parameters directly. Instead it is more intuitive to specify a density (attenuation coefficient) and scattering color (albedo). As long as the scattering color is in the range 0..1, the volume shader will be energy conserving. Values outside the range may be used for non-physically real scattering effects.

## Volume Density and Scattering Color

```
shader_evaluate
{
    const float density = AiShaderEvalParamFlt(p_density);
    const AtRGB scattering_color = AiShaderEvalParamRGB(p_scattering_color);
    const float anisotropy = AiShaderEvalParamFlt(p_anisotropy);

    const AtRGB absorption = density * (1 - scattering_color);
    const AtRGB scattering = density * scattering_color;
    const AtRGB emission = AI_RGB_BLACK;

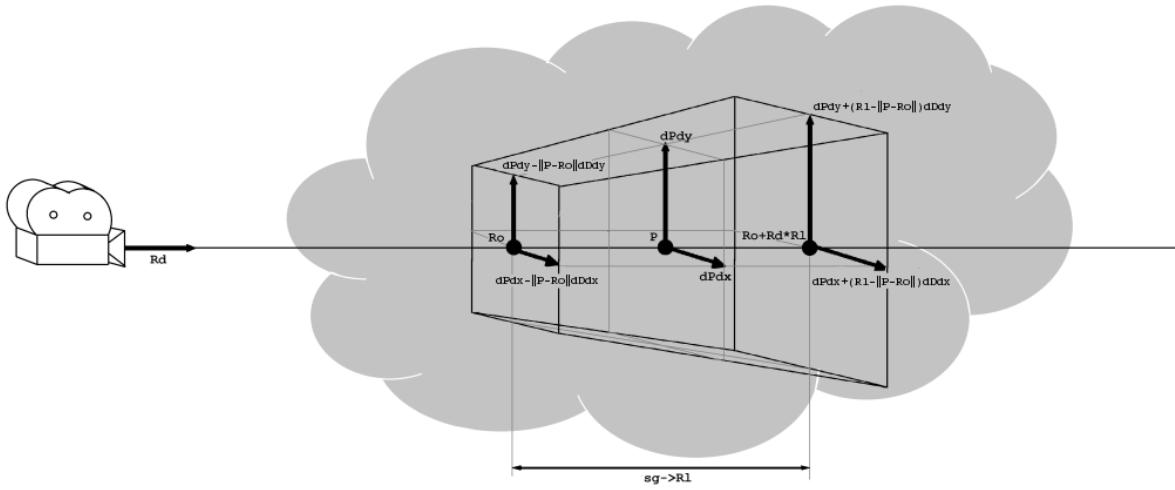
    sg->out.CLOSURE() = AiClosureVolumeHenyeyGreenstein(sg,
        absorption, scattering, emission, anisotropy);
}
```

## Shader Globals

Volumetric shaders can expect the following shader globals to be readily available for use:

- **sg->Rd** : direction of ray traversing volume
- **sg->Ro** : position indicating the beginning of the segment of the volume that is being evaluated
- **sg->Rl** : length of the segment of the volume that is being evaluated
- **sg->P** : world-space sample position within the volume segment that is being evaluated
- **sg->Po** : object-space sample position within the volume segment that is being evaluated
- **sg->M** : object to world space transform matrix
- **sg->Minv** : world to object space transform matrix
- **sg->Op** : pointer to volume's container shape
- **sg->dPdx**: rate of change of sg->P as samples move along the x axis of the image plane
- **sg->dPdy**: rate of change of sg->P as samples move along the y axis of the image plane

Here is a diagram of these shader globals:



## Example Shader

Here is a small example of a volume shade that places a heterogeneous volume, modulated by a procedural noise effect and a spherical bounds, in the object space of the volume's container shape:

## Example Volume Shader

```
shader_evaluate
{
    const AtVector c = AiShaderEvalParamVec(p_position);
    const float r = AiShaderEvalParamFlt(p_radius);
    const int octaves = AiShaderEvalParamInt(p_octaves);
    const float scale = AiShaderEvalParamFlt(p_scale);

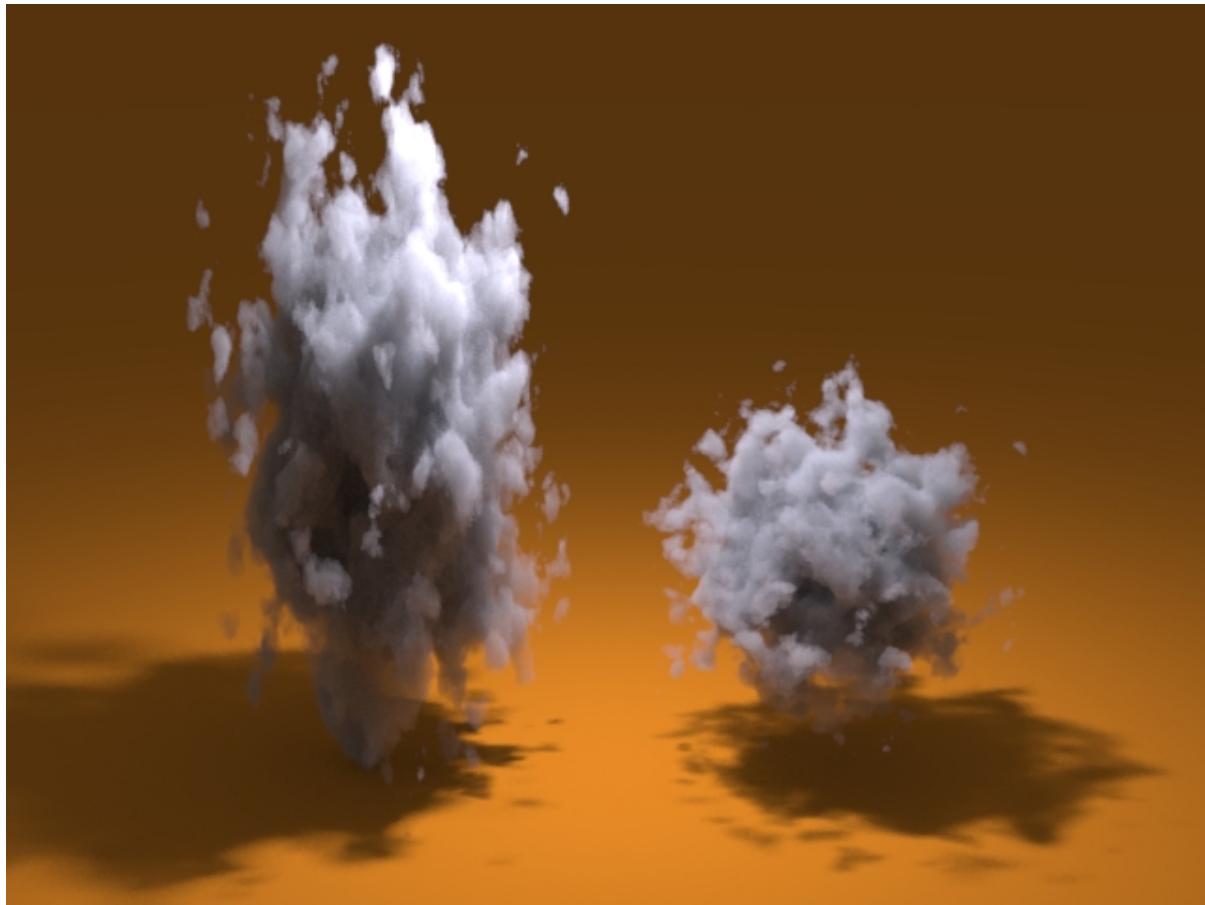
    const float density = 25.f;

    const AtVector p = sg->Ro;
    const float rel_dist = AiV3Length(p - c) / r;           // in [ 0,1]
    const float threshold = rel_dist * 2 - 1;               // in [-1,1]
    float noise = AiNoise3(p * scale, octaves, 0, 1.92f); // in [-1,1]
    noise = noise > threshold ? density : 0;

    const AtRGB absorption = AI_RGB_BLACK;
    const AtRGB scattering = AtRGB(noise);
    const AtRGB emission = AI_RGB_BLACK;

    sg->out.CLOSURE() = AiClosureVolumeHenyeyGreenstein(sg,
        absorption, scattering, emission);
}
```

When applied to two different container shapes, the left shape with a vertical scaling that is twice that of the right shape, the results from this shader could look like this: (remember to set **step\_size** on the bounding shapes, and **options.GI\_volume\_samples** if you want indirect lighting on the volume)



Note that, because volume shaders can be called dozens or even hundreds of times per ray, the **shader\_evaluate** code must be as efficient as possible. Even harmless looking calls like **AiShaderEvalParam\*** can add significant overhead if used excessively so you will want to keep those to a minimum or precompute them in **node\_update** when possible.

# Textures

- [maketx](#)
- [Filename Tokens](#)

When using Arnold it is best to use a tiled mip-mapped texture format such as .exr or .tx that has been created using maketx.

.tx textures are:

1. Tiled (usually the tiles are 64x64 pixels).
2. Mip-mapped.

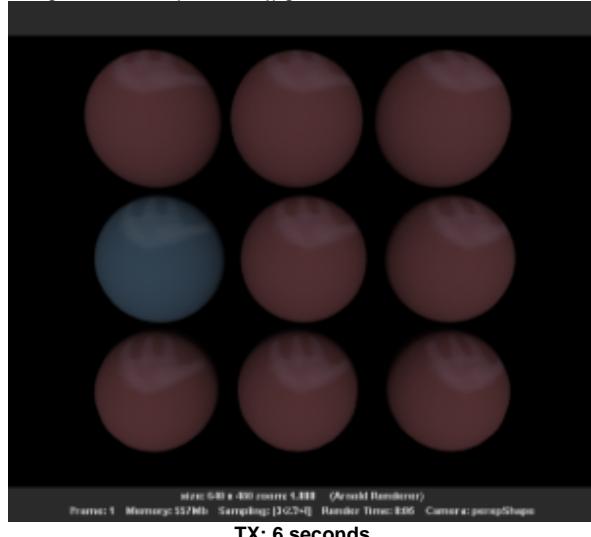
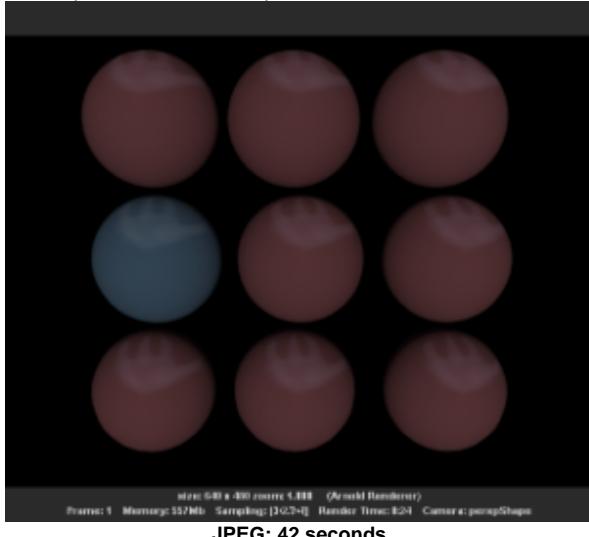
If you already have tiled and mip-mapped exr's that have been created by another renderer, you won't need to convert those files to .tx

Due to (1), Arnold's texture system can load one tile at a time, as needed, rather than having to wastefully load the entire texture map in memory. This can result in faster texture load times, as texels that will never be seen in the rendered image will not even be loaded. In addition to the speed improvement, only the most recently used tiles are kept in memory, in a texture cache of default size 512 mb (can be tuned via options.texture\_max\_memory\_MB). Tiles that have not been used in a long time are simply discarded from memory, to make space for new tiles. Arnold will never use more than 512 mb, even if you use hundreds, or thousands of 4k and 8k images. But then, if you only use a handful of 1k textures, this will not matter.

Due to (2), the textures are anti-aliased, even at low AA sample settings. Neither of these are possible with JPEG or other untiled/unmipped formats (unless you tell Arnold to auto-tile and auto-mip the textures for you, but this is very inefficient because it has to be done per rendered frame, rather than a one-time pre-process with maketx).

It should be worth pointing out that .tx files are basically .exr or .tif files that have been renamed to .tx. This means .tx files can be read by image editors, although you may need to rename the file to .exr or .tif so that the image editor will load it. However, .tx files have a few extra custom attributes set that are not normally included in .exr/.tif files which can make rendering even faster. For instance, they will include a hash so that if you try to load two different files that contain the same data, Arnold only needs to load this data once.

The example below shows a speed increase of seven times when using .tx files compared to .jpg files:



## Manually Generate TX Textures

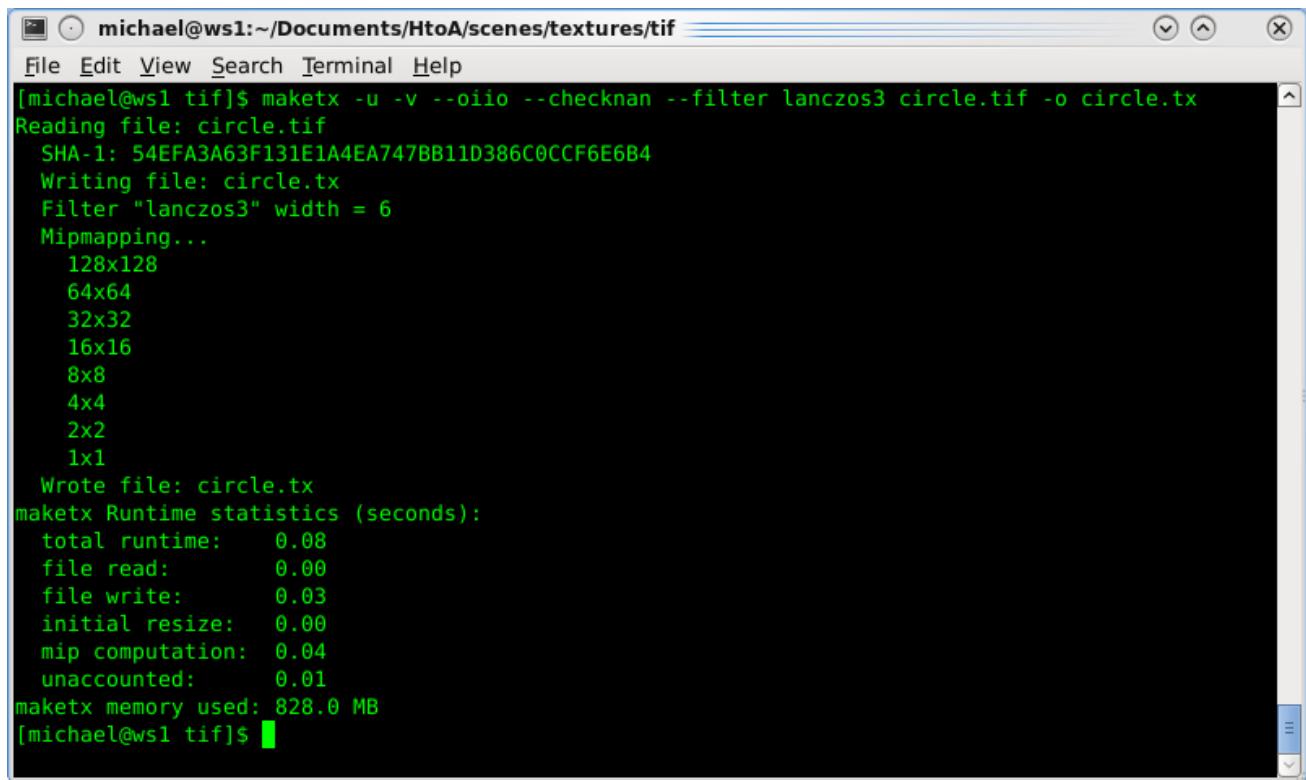
The process is quite simple. You will need the maketx.exe utility, the texture that needs to be converted and a dos shell. Below is an example of how you would convert a .tif file to a .tx file.

C:\Windows\system32\cmd.exe  
Microsoft Windows [Version 6.1.7601]  
Copyright © 2009 Microsoft Corporation. All rights reserved.  
  
C:\Users\Documents\projects\MakeTx\sourceimages>maketx --oii0 C:\Users\MakeTx\sourceimages\texture.tif  
  
Folder which contains the texture that needs to be converted.  
The maketx.exe utility has also been copied to this folder. Makebx command with the optimize '--oii0 flag' and the pathname of the .tif file that needs to be converted.

## maketx

### *synopsis*

Maketx is a command-line utility to convert images to tiled, MIP-mapped textures, similar to txmake in Pixar Renderman. It is part of OpenImageIO (<http://www.openimageio.org>) and was developed by Larry Gritz at Sony Pictures Imageworks.



```
[michael@ws1 tif]$ maketx -u -v --oii0 --checknan --filter lanczos3 circle.tif -o circle.tx
Reading file: circle.tif
SHA-1: 54EFA3A63F131E1A4EA747BB11D386C0CCF6E6B4
Writing file: circle.tx
Filter "lanczos3" width = 6
Mipmapping...
  128x128
  64x64
  32x32
  16x16
  8x8
  4x4
  2x2
  1x1
Wrote file: circle.tx
maketx Runtime statistics (seconds):
  total runtime: 0.08
  file read: 0.00
  file write: 0.03
  initial resize: 0.00
  mip computation: 0.04
  unaccounted: 0.01
maketx memory used: 828.0 MB
[michael@ws1 tif]$
```

The maketx command with verbosity, converting from a tif to a tx.

### *usage*

```
maketx [options] file...
```

```
eg: maketx -v -u --oii0 --checknan --filter lanczos3 path/to/fileIn.tif -o path/to/fileOut.tx
```

When pre-processing your texture files with **maketx** make sure you add the **--oii0** flag which will generate TX files optimized for Arnold that can render even faster by forcing the tile size to be optimal for OpenImageIO (64x64). It also stores channels interleaved (RGBARGBA...) making them faster to read and enables constant color detection: if the image is a constant color it will convert it to a tiny single color .tx file. Additional metadata and an image hash (fingerprint) are embedded by default. The fingerprint lets OIIO detect duplicate images (not tiles) when opening new files. Generally using maketx will result in better texture I/O performance than using txmake.

Note also that if you need your textures to stay compatible with renderman, there is a **--prman** switch that will ensure the tile size matches what renderman expects.

If the output color space is not the same bit depth as input color space, then the data format needs to be set to the proper bit depth using the **-d** option.

The best workflow is to always generate .tx files using maketx as a last step before rendering, and use those .tx files just for rendering. This will ensure the best performance. Also ensure that your textures are in linear color space before converting them with maketx. No information is lost when converting High Dynamic Range images to the .tx format.

## parameters

flag	description
--help	Print help message
-v	Verbose status messages
-o %s	Output filename
--threads %d	Number of threads (default: #cores)
-u	Update mode
--format %s	Specify output file format (default: guess from extension)
--nchannels %d	Specify the number of output image channels.
-d %s	Set the output data format to one of: uint8, sint8, uint16, sint16, half, float
--tile %d %d	Specify tile size
--separate	Use planarconfig separate (default: contiguous)
--fov %f	Field of view for envcube/shadcube/twofish
--fovcot %f	Override the frame aspect ratio. Default is width/height.
--wrap %s	Specify wrap mode (black, clamp, periodic, mirror)
--swrap %s	Specify s wrap mode separately
--twrap %s	Specify t wrap mode separately
--resize	Resize textures to power of 2 (default: no)
--noresize	Do not resize textures to power of 2 (deprecated)
--filter %s	Select filter for resizing (choices: box triangle gaussian catrom blackman-harris sinc lanczos3 radial-lanczos3 mitchell bspline, disk, default=box)
--nomipmap	Do not make multiple MIP-map levels
--checknan	Check for NaN/Inf values (abort if found).
--Mcamera %f %f %f %f %f %f %f %f %f %f %f %f %f %f	Set the camera matrix
--Mscreen %f %f %f %f %f %f %f %f %f %f %f %f %f %f	Set the camera matrix
--hash	Embed SHA-1 hash of pixels in the header (deprecated. hashes are always computed).
--prman-metadata	Add prman specific metadata
--constant-color-detect	Create 1-tile textures from constant color inputs
--monochrome-detect	Create 1-channel textures from monochrome inputs
--opaque-detect	Drop alpha channel that is always 1.0
--stats	Print runtime statistics
--mipimage %s	Specify an individual MIP level
<b>Basic modes (default is plain texture)</b>	
--shadow	Create shadow map
--envlatlong	Create lat/long environment map
--envcube	Create cubic env map (file order: px, nx, py, ny, pz, nz) (UNIMP)
<b>Color Management Options</b>	
--colorconvert %s %s	Apply a color space conversion to the image. If the output color space is not the same bit depth as input color space, it is your responsibility to set the data format to the proper bit depth using the -d option. (choices: linear, sRGB, Rec709)
--unpremult	Unpremultiply before color conversion, then premultiply after the color conversion. You'll probably want to use this flag if your image contains an alpha channel.
<b>Configuration Presets</b>	
--oiio	Use OIIO-optimized settings for tile size, planarconfig, metadata, and constant-color optimizations.
--prman	Use PRMan-safe settings for tile size, planarconfig, and metadata.

## Filename Tokens

Arnold supports the following texture tokens, including both the `<tile>` (Mudbox) and `<udim>` (Mari) tokens.

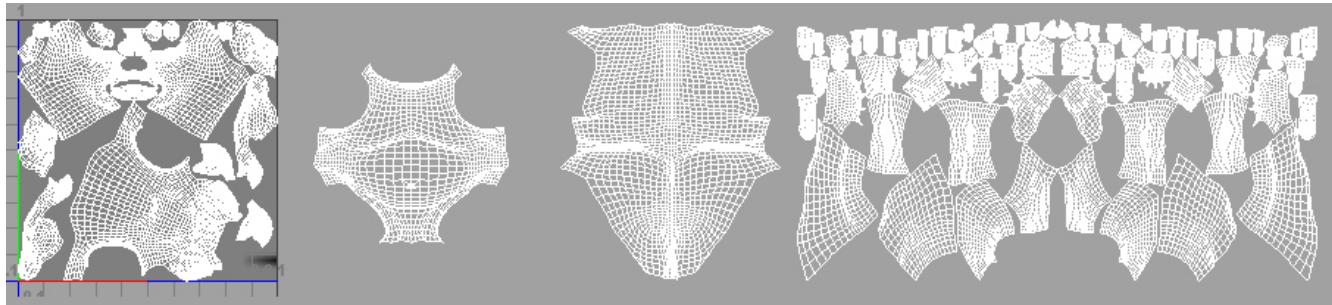
- `<attr:>`
- `<tile>`
- `<udim>`

## <attr:>

The <attr:> token enables you to apply different texture maps based on the name of the attribute that is assigned to the mesh. It is 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.

## <tile>

When using the <tile> token, multiple textures can be used to cover a model, rather than a single, much larger, texture. It is the notation at which it is exported in UV texture space. This starts at 1001 (0,0) and continues in U for another 10 offsets. It then progresses onto the next row of V, meaning for every 10 in U it moves 1 in V.

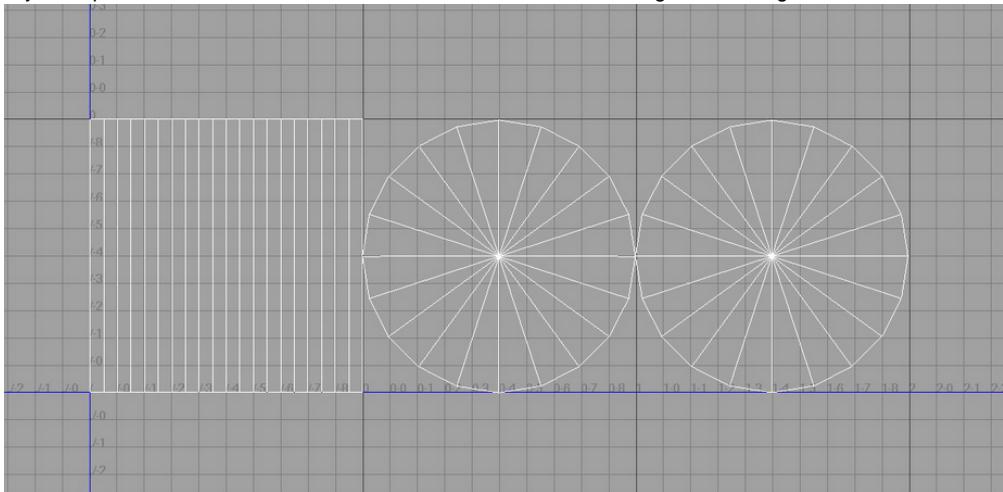


UV's should be arranged into separate UV tiles in order to maximize texture space

You can use a single File texture to open textures composed of multiple images that correspond to the grid tiles in your UV layout. You can, therefore, open and render high-resolution textures produced by 3D painting applications such as Mudbox, and is a better alternative to using UVsets.

## <udim>

UDIM values are a way of representing the integer coordinates of a texture, from the coordinates of its bottom-left corner in UV space. This way, multiple textures can be used to cover a model, rather than a single, much larger, texture.



A cylinder with its side in  $u0,v0$ , base in  $u0,v1$ , and top in  $u0,v2$

It is a 4 digit number starting at 1001, created using the formula,  $1001 + u + (10 * v)$

In the example above the side would be 1001, base 1002, and top 1003.

To use this method in your file inputs, use the <udim> tag in the file name, which is then replaced with the appropriate number.

`texture.<udim>.exr` will be read as `texture.1001.exr` when looked up for rendering.

Check out Ben Neall's [page](#) for a clear description of UDIM numbering.

# Third Party Shaders

## Installation

The recommended way to install third-party shaders is to use the ARNOLD\_PLUGIN\_PATH environment variable. For example, set the ARNOLD\_PLUGIN\_PATH environment variable like so:

```
ARNOLD_PLUGIN_PATH=C:\solidangle\alShaders\bin
```

There are several third-party shader resources available for the Arnold plugins, as listed on the Solid Angle [website](#).

These third-party shaders, extensions, and integrations are not developed or supported by Solid Angle but are provided for your convenience.

## Legal

- [End User License Agreement](#)

# End User License Agreement

This End User License Agreement ("EULA") is a legal agreement between you, an individual or other legal entity ("End User") and Solid Angle Limited, a company registered in England and Wales with registered number 8096594 ("Solid Angle"), which sets forth the terms and conditions of the license to use the Arnold software, plug-ins and accompanying documentation ("Arnold Software").

By downloading, installing, copying, accessing or otherwise using the Arnold Software, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not download, install or use the Arnold Software.

The attention of the End User is particularly drawn to clauses 4 and of this EULA regarding the extent of the warranty Solid Angle gives to the End User and the extent to which it accepts liability to the End User.

The Arnold Software is licensed and not sold. In return for acquiring a license to use the Arnold Software the End User agrees as follows:

## 1. License Grant

In consideration of the payment in full of the license fee, Solid Angle grants the End User a limited, non-exclusive and non-transferable license: ( a ) to use the Arnold Software; ( b ) to install the Arnold Software on a network server and use that number of copies of the Arnold Software as have been licensed and paid for at any one time; ( c ) to copy the Arnold Software solely for the purpose of installing it on the End User's computers and for backup storage.

The creation of multiple instances of the Arnold Software on a single computer with multiple processors shall be deemed a single licensed copy of the Arnold Software for the purposes of calculating the number of copies used pursuant to this license.

The use of the Arnold Software at any facility owned by the End User regardless of geographic location is permitted under this license.

## 2. Restrictions on Use

The End User may only use the Arnold Software for authorised purposes in relation to its own business and not to provide any service to third parties, and may not: ( a ) assign, sublicense, loan, sell, distribute, share, transfer, pledge, lease or rent the Arnold Software or End User's rights hereunder to others without the written permission of Solid Angle; ( b ) (except to the extent permitted by English law) reverse engineer, decompile, disassemble, alter or otherwise attempt to discover the source code of the Arnold Software; ( c ) circumvent the license management software; ( d ) adapt, modify, translate or create derivative works based on the Arnold Software.

The Arnold Software will operate in evaluation mode with a watermarked output if a valid license key is not installed. The use of the watermarked output for commercial use is strictly prohibited.

Licenses that are used for education purposes or have been purchased at an educational discount may only be used by the End User for training and instruction and for no other purpose.

The End User may not export or re-export the Arnold Software to any country or entity subject to United Kingdom export restrictions.

## 3. Ownership

The End User agrees that the Arnold Software and all intellectual property rights shall remain the sole property of Solid Angle or its licensor. The license granted in this EULA is not a sale of the Arnold Software and this EULA does not give the End User any rights to patents, copyrights, trademarks, service marks, trade secrets, confidential information or any other rights or license with respect to the Arnold Software. The End User agrees to hold the Arnold Software in confidence and to take reasonable measures to prevent unauthorised copying or disclosure. The End User shall not remove or modify any copyright symbols, trademarks, labels and property notices on any and all copies of the Arnold Software.

## 4. Limited Warranty

Solid Angle warrants that:

(a) the Arnold Software will, when properly used and on an operating system for which it was designed, perform substantially in accordance with the functions described in the accompanying documentation; and (b) that the accompanying documentation correctly describes the operation of the Arnold Software in all material respects,

in each case for a period of 90 days from the date that the End User downloads the Arnold Software (Warranty Period).

If, within the Warranty Period, the End User notifies Solid Angle in writing of any defect or fault in the Arnold Software as a result of which it fails to perform substantially in accordance with the accompanying documentation, Solid Angle will, at its sole option, either repair or replace the Arnold Software, provided that the End User makes available all the information that may be necessary to help Solid Angle to remedy the defect or fault, including sufficient information to enable it to recreate the defect or fault.

The warranty given by Solid Angle does not apply if the defect or fault in the Arnold Software results from any amendment or attempted amendment of the Arnold Software by any person other than Solid Angle or if the defect or fault in the Arnold Software results from use of the Arnold Software in contravention of the terms of this EULA.

The End User acknowledges that the Arnold Software has not been developed to meet the End User's individual requirements and that it is therefore the End User's responsibility to ensure that the facilities and functionality of the Arnold Software meet such requirements. Solid

Angle does not warrant that the Arnold Software will meet the End User's requirements or that the operation of the Arnold Software will be uninterrupted or error free, or that defects in the Arnold Software will be corrected. Solid Angle does not warrant the correctness or accuracy of the results obtained from the use of the Arnold Software.

Except as expressly stated in this EULA, there are no conditions, warranties, representations or other terms, express or implied, that are binding on Solid Angle. Any condition, warranty, representation or other term concerning the supply of the Arnold Software which might otherwise be implied into, or incorporated in, this EULA whether by statute, common law or otherwise, is excluded to the fullest extent permitted by law.

## **5. Limited Liability**

Nothing in this EULA shall limit or exclude Solid Angle's liability for death or personal injury resulting from its negligence, for fraud or fraudulent misrepresentation or for any other liability that cannot be excluded or limited by English law.

In no event shall Solid Angle or its suppliers be liable to the End User for any third party claims or any consequential, incidental, punitive, special, or indirect damages of any kind (including but not limited to damages resulting from loss of use, loss or corruption of data, loss of profits, loss of goodwill, loss of business or other loss or damage suffered) arising out of or relating to the use of the Arnold Software, regardless of whether such loss was reasonably foreseeable or Solid Angle had been advised of the possibility of such damages. In no event will Solid Angle's liability for any claim, whether in contract, tort (including negligence), misrepresentation (whether innocent or negligent), negligent misstatement, breach of statutory duty or any other theory of liability, exceed the license fee paid by the End User.

## **6. Events Outside our Control**

We will not be liable or responsible for any failure to perform, or delay in performance of, any of our obligations under this EULA that is caused by an Event Outside Our Control, as defined below.

An Event Outside Our Control means any act or event beyond our reasonable control, including without limitation failure of public or private telecommunications networks.

If an Event Outside Our Control takes place that affects the performance of our obligations under this EULA our obligations under this EULA will be suspended and the time for performance of our obligations will be extended for the duration of the Event Outside Our Control.

## **7. Term and Termination**

This EULA shall remain in effect until terminated.

Solid Angle may terminate this EULA, and the license it grants, immediately if the End User breaches any term of this EULA. On termination by Solid Angle, the End User agrees to immediately stop using the Arnold Software, permanently delete the Arnold Software and all copies and within five (5) days thereafter provide Solid Angle with written confirmation that it has complied with these actions.

The End User may terminate this EULA, and the license it grants, at any time by permanently deleting the Arnold Software and all copies and notifying Solid Angle.

## **8. Law and Jurisdiction**

This EULA and any dispute or claim arising out of or in connection with it or its subject matter or formation (including non-contractual disputes or claims) shall be governed by English law and the parties irrevocably submit to the non-exclusive jurisdiction of the English Courts.

# System Requirements

In general, Arnold is going to work on pretty much any **64-bit** system where Houdini, Maya, Cinema 4D, 3ds Max, Katana, or Softimage works. However, there are some minimum OS requirements:

- OSX 10.8 or later
- Windows 7 or later, with the Visual Studio 2015 redistributable
- Linux with at least glibc 2.12 and libstdc++ 3.4.13 (gcc 4.4.7). This is equivalent to RHEL/CentOS 6
- CPUs need to support the SSE4.1 instruction set

Other than that, Arnold works on Intel and AMD processors, and for rendering, has **no graphic card or GPU requirements**, since Arnold is a CPU-based renderer.

# Arnold Denoiser (*Noice*)

The Arnold denoiser (*Noice*) is a stand-alone program post process denoiser executable. It works on EXR files with multiple layers and outputs an EXR file with the denoised layers. It takes into account multiple frames and multiple light AOVs. It requires variance information for all AOVs and optionally uses normal, depth and albedo. Like most denoisers, it considers a neighborhood around the current pixel and looks for similar neighborhoods inside a given search area to blend together.

## Inputs

### Required Input Layers

The required absolute minimum inputs are the following:

- main layer to be denoised (typically RGBA also known as "beauty")
- variance layer that specifies the per pixel variance

### Optional Feature Layers

These optional layers are used to guide denoising and will greatly improve the result:

- Normal (N)
- Depth (Z)
- Albedo (diffuse\_albedo)

### Optional Light AOVs to be Co-Denoised

Additional light AOVs can be present in the original EXR. If requested (-l light\_aov\_name) these will be co-denoised with the beauty layer. This main denoising effort will be spent on beauty, so in certain areas the light AOVs might have increased noise.

### Optional Additional Frames

Multiple additional frames can be specified to increase temporal stability (for instance -i render.002.exr -i render.001.exr -i render.003.exr)

## Outputs

The output file will include the denoised layers with the same names as the original layers.

## Arguments

### Pixel Neighborhood Patch Radius

This increases the softness of the denoising (while always preserving the features). The default is 3 (on the big side), but a low value would be maybe 0 or 1, middle 3 and high 5.

For every pixel **noise** will consider its neighborhood patch and look for other pixels with similar neighborhood patches. The radius of this neighborhood can be controlled with the **-patchradius** (or **-pr**) command line argument. The default value is set to 3, which gives a 7x7 square neighborhood.



**Without denoising**

**1**

**3 (default)**

### Pixel Search Radius

This is the area over which similar neighborhoods are found. The higher the better, but it will increase the cost of denoising. For every pixel `noice` will search a square area with a radius set with the command line argument `-searchradius (-sr)`. The bigger this area the bigger the denoising stability and the higher the chance that similar neighborhoods to be considered will be found. The default value is 9, which gives a 19x19 square neighborhood. Setting it to 21 (a search window of 42 x 42) will look over a pixel area equivalent to loading 5 frames.



**6**

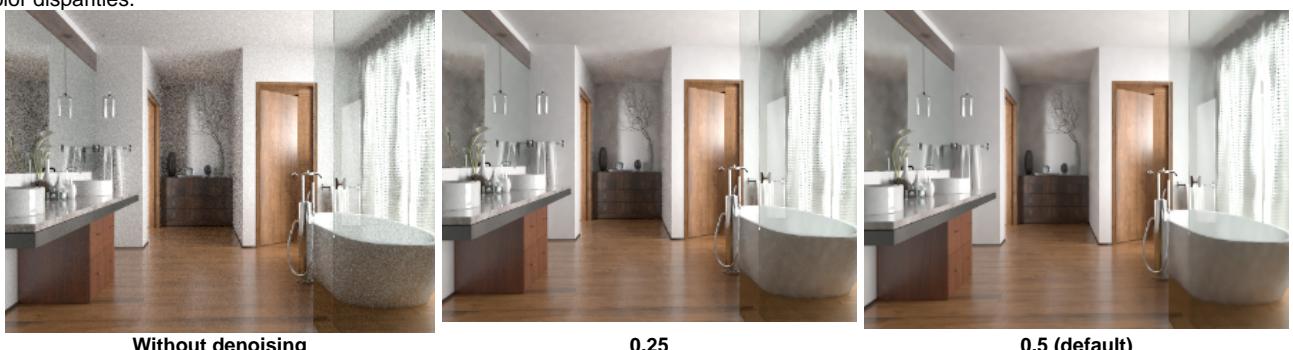
**9 (default)**

**18**

### Variance

The strength of the filter is determined by the variance parameter, the higher the variance the more forceful the denoising will be. For variance maybe 0.25 / 0.5 (the default) / 0.75 are good low/min/max values.

How aggressive `noice` is in removing noise can be controlled by setting a variance threshold with the command line argument `-variance (-v)`. The default value is 0.5, higher values will make the denoising more aggressive by considering similar neighborhoods that have bigger color disparities.



**Without denoising**

**0.25**

**0.5 (default)**

### Input Files

Input files to `noice` can be specified with the command line argument `-input (-i)`. Multiple frames can also be specified around the frame to be denoised in the following way:

```
-i <current_frame> -i <additional frames in order>
-i render.0010.exr -i render.0008.exr -i render.0009.exr -i render.0011.exr -i render.0012.exr
```

### Light AOVs

Additional light AOVs to be co-denoised can be specified with the command line argument `-aov (-l)`, for instance:

```
-l back_lights -l RGBA -l key_light
```

### Output File

The output file can be specified with the command line argument `-output (-o)`.

## Ass File Example

These additional AOV are optional, but the variance info for RGBA and additional AOVs to be denoised is not. The denoiser will work much better if N,Z and diffuse\_albedo are provided:

```
outputs 7 1 STRING
"RGBA RGBA filter driver"
"RGB RGB variance driver RGB_variance"
"LPE_name RGBA filter driver"
"LPE_name RGB variance driver LPE_name_variance"
"diffuse_albedo RGB filter driver albedo"
"N RGB defaultArnoldFilter@gaussian_filter driver"
"Z RGB filter driver"
```

Notice that the AOVs that have the same source also have an EXR layer name specified at the end. The name of the layers is not important, but appending `_variance` seems like the clearer way to handle this.

The filters should be set up like this (here we use `gaussian_filter` but it could be any other filter as long as it is matched in `variance_filter.filter_weights`)

```
<filter_type>_filter
{
    name filter
    width <radius>
}

variance_filter
{
    name variance
    filter_weights <filter_type>
    width <radius>
    scalar_mode false
}
```

## FAQs

## Arnold 5.0 FAQ

Below are some commonly asked questions with regards to changes in Arnold 5.

### Where is the Standard shader?

▼ [Click here to expand...](#)

The old **Standard** shader has been replaced with the **Standard Surface** shader. The old **Standard** shader is still shipped with Arnold as a legacy shader, for compatibility with old scenes. However, it will be removed in a future release.

### Is the Standard Surface shader energy conserving?

▼ [Click here to expand...](#)

If all of the individual weights and colors are less than or equal to 1.0, then the **Standard Surface** shader is energy conserving. Unlike the old Standard shader, you don't need to worry about the sum of weights being less than 1.0 or manually enable Fresnel, which is always enabled.

### Where is 'Refraction Roughness' in the Standard Surface shader?

▼ [Click here to expand...](#)

The **Specular Roughness** affects both specular reflection and refraction. There is also an **Extra Roughness** parameter in **Transmission** to add some additional roughness for refraction. You can, however, use **Coat** to create a rough reflection layer over a sharp refraction.

### Where is the specular GGX microfacet distribution?

▼ [Click here to expand...](#)

GGX is now the default model for the **Standard Surface** shader.

### How can I control the index of refraction for transparent materials?

▼ [Click here to expand...](#)

You can use the **Specular IOR** attribute in the same way to handle transparent materials such as glass, diamond, etc.

### How can I control Backlighting/Translucency in the Standard Surface shader?

▼ [Click here to expand...](#)

Enabling **Thin Walled** and setting **Subsurface Weight** to, for example, 0.5 to have half the light reflected and half transmitted.

### IOR and Metalness appear to do the same thing. Why use one over the other?

▼ [Click here to expand...](#)

Using very high IOR values can look quite similar to metalness. It looks the same if you set base color to the specular color and the specular color to black. The difference is that you get an extra reflection at the edges, with the specular color controlling the edge tint. The metal Fresnel works the same as in the **Complex IOR** shader, with the artistic parameters.

### Where is the Skin shader?

▼ [Click here to expand...](#)

You should use the **Subsurface** parameter of the **Standard Surface** shader. It uses the high-quality empirical SSS profile, which preserves sharp geometric and bump detail without the need for three layers of varying depth, as the old skin shader did.

### Where is the Hair shader?

▼ [Click here to expand...](#)

Use the new **Standard Hair** shader instead. The old **Hair** shader is shipped with Arnold as a legacy shader, for compatibility with old scenes. However, it will be removed in a future release.

### Where have the UV parameters gone?

▼ [Click here to expand...](#)

The hair shader does not have a "uvset" parameter at all. Rather all image and procedural texture nodes have a "uvset" parameter instead so that they can pick up the appropriate texture coordinates there.

### Where is the Volume Collector shader?

▼ [Click here to expand...](#)

Use the new **Standard Volume** shader instead. The old **Volume Collector** shader is shipped with Arnold as a legacy shader, for compatibility with old scenes. However, it will be removed in a future release.

## **How can I use OSL shaders?**

▼ [Click here to expand...](#)

Information about OSL shaders can be found [here](#).

## **Where is the Shadow Catcher shader?**

▼ [Click here to expand...](#)

The Shadow Catcher shader is now called the *Shadow Matte* shader.

### **ASS Files**

Although the syntax of the file format hasn't changed at all, .ass files generated with Arnold 4 may not render correctly with Arnold 5 and so should be avoided. Likewise, .ass files generated with Arnold 5 may not render correctly with Arnold 4.

### **Sky Shader**

The sky shader is deprecated and will be removed from a future release. You should instead use the visibility settings in the *Skydome Light*.

### **Skydome Light**

The **Camera** visibility in the skydome light replaces the need for creating a sky shader and carefully setting its visibility attributes to avoid double counting. The Skydome light shows up in the indirect AOVs, not the direct. It can be changed to the direct light AOV by setting the light's AOV to "default" to place it in the default light group, or another name for some other light group.

## General FAQ

### **How can I do glossy reflections using the Standard Surface shader?**

▼ [Click here to expand...](#)

In the specular section, adjust the "Roughness" parameter to control how blurry your reflections are. The lower the value the sharper the reflection. In the limit, a value of 0 will give you perfectly sharp mirror reflection.

### **Fireflies, why do they appear and how can they be avoided?**

▼ [Click here to expand...](#)

Certain scenes/configurations suffer from a form of sampling noise commonly referred to as "spike noise", or "fireflies": isolated, super bright pixels that jump around from frame to frame in an animation. This is especially noticeable in recursive specular reflections involving both diffuse and sharp specular surfaces. This noise is very difficult to remove by just increasing the number of samples in the renderer. There are several ways to fix the noise.

- Make the objects with the sharp specular surfaces invisible to specular rays. This can be done by disabling the specular flag in the object's visibility parameters.
- Use a ray\_switch shader in the objects, with an appropriately modified shader in the "glossy" slot - for example, a shader returning black, or perhaps a shader with a bigger specular\_roughness value.

### **Are caustics possible?**

▼ [Click here to expand...](#)

Given that Arnold uses uni-directional path tracing, "soft" caustics originating at specular surfaces are perfectly possible, as well as caustics coming from big sources of indirect light. The caustics switches in the standard shader mean that you can tell the diffuse GI rays to "see" the mirror reflection, specular reflection and refraction from the shader of the surfaces that are hit by them. By default only the direct and indirect diffuse rays are seen by GI rays. On the other hand, "hard" caustics emanating from spatially-small but bright direct light sources, for example caustics from a spotlight through a glass of cognac, are currently not possible. One possible workaround to render caustics would be to light the scene with light-emitting geometry where you set the values for emission really high (20-100) and play with the size of the emitter. However you would have to use really high sample settings to avoid grain. Other renderers more or less easily achieve hard caustics with the photon mapping technique. At Solid Angle we dislike photon mapping because it's a biased, memory sucker technique that is prone to artifacts, blurring, obscure settings, doesn't scale well with complex scenes and doesn't work well with interactivity/IPR.

### **Bump mapping is not working when connecting images to a Bump3D node**

▼ [Click here to expand...](#)

Bump3D works by evaluating the bump shader (and image in this case) at different points ( $P + \epsilon$ ,  $P + \epsilon$ ,  $P + \epsilon$ ). Since the only thing displaced is the point, the UV coordinates will be the same in the different lookups. These will give the same texel in the image and result in no perturbation of the normal. You should use Bump2D for images.

### **How to get rid of noise**

▼ [Click here to expand...](#)

Computationally, the efficient way to get rid of noise is to go from the bottom up (going from the particular to the general). We would increase the sampling of individual lights first, then the GI and specular samples. Finally, the AA samples, which acts as a global multiplier of the whole sampling process. However, the only way to improve the quality and reduce the noise of motion blur and depth of field is to increase the AA samples. In this case, this AA increase allows you to decrease the other sampling rates (GI/specular/light) to compensate. To sum up, you will get almost the same render time with AA=1 and GI=9 than with AA=9 and GI=1, but at AA=9 you will have much better motion blur and depth of field quality. More information can be found [here](#) and a workflow [here](#).

### **What does `min\_pixel\_width` do for curves?**

▼ [Click here to expand...](#)

`min_pixel_width` is a threshold parameter that limits how thin the curves shapes can become with respect to the pixel size. So if you were to set `min_pixel_width` to a value of 1 pixel, no matter how thin or far away from the camera that the curves are they will be "thickened" enough so that they appear 1 pixel wide on screen. Wider strands are easier to sample so they will tend to show much less aliasing artifacts.

The problem with simply thickening curves like this is that they will start to take on a wiry or straw-like appearance because they are much wider and blocking much more of the background than they should be.  
A well-established method of giving a softer look to thick strands is to map their opacity along their length, the internal user data "geo\_opacity" is an automatic way to do this.

The value of the "geo\_opacity" UData field for strands that are already thicker on screen than the `min_pixel_width` threshold will always be 1.0, but strands that had to be thickened to meet the `min_pixel_width` threshold will get lower geo\_opacity values in proportion to the

amount of thickening. If a shader reads this value and uses it to scale its out\_opacity result, then the thickening of the curves will be properly compensated and thin curves will retain their soft appearance. The result is not exactly the same as the curves without thickening, but on average the appearance is very similar and can be much easier for the raytracer to sample.

In practice a min\_pixel\_width setting of 1.0 is probably too high, making the strands look too soft and the difference between using and not using the technique quite noticeable. You will usually get better results with values in the 0.1-0.5 pixel range. Also, the higher the min\_pixel\_width, the slower to render, because the renderer will make the hairs more transparent to compensate for their increased thickness. For example, a value of 0.25 means that the hair geometry will never be bigger than 1/4 the size of the pixel, so you can get good antialiasing with AA=4 samples. A value of 0.125 (or 1/8) will need at least AA=8 to get good antialiasing etc.

### **What does autobump do for polymeshes?**

▼ [Click here to expand...](#)

When autobump is enabled, Arnold makes a copy of all of the vertices of a mesh prior to displacement (let's call that the "reference" mesh, or Pref). Prior to shading at some surface point on the displaced surface P, the equivalent Pref for that point is found on the non-displaced surface and the displacement shader is evaluated there (at Pref) to estimate what would be the equivalent normal at P if we had subdivided the polymesh at an insanely high tessellation rate.

The main difference between Arnold's autobump and using the displacement shader for bump mapping (say with the bump2d node) is that autobump has access to Pref whereas bump2d does not and would be executing the displacement shader on already-displaced points which could "compound" the displacement amounts (if that makes any sense).

The only extra storage is for copying P prior to displacement. There is no analysis of the displacement map; Arnold displaces vertices purely based on where they "land" in the displacement map (or procedural) regardless of whether it happens to "hit" a high-frequency spike or not..

### **Autobump does not work**

▼ [Click here to expand...](#)

The autobump algorithm needs UV coordinates to compute surface tangents. Make sure your poly mesh has a UV set applied.

### **How is transparency handled?**

▼ [Click here to expand...](#)

Arnold has two different ways of calculating transparency, transmission, and opacity. They are different ray types and thus have different controls in the standard surface shader as well as in the render options. You must disable 'opaque' for the mesh that has been assigned the standard surface material.

### **How do I work in Linear colorspace with Arnold?**

▼ [Click here to expand...](#)

Arnold automatically works in linear color space, all light and shader colors are assumed to be in the linear rendering space. For textures and render output, color spaces may be specified to convert to and from the linear rendering space.

### **What are .tx files?**

▼ [Click here to expand...](#)

.tx are just tiled+mipmapped tiff files.

The 'maketx' utility part of OpenImageIO (or the 'txmake' utility shipped with RenderMan) can convert any image file into a TX file. It gets slightly more confusing because it is also common to rename .exr files to .tx, since OpenEXR also supports tiling and mipmapping (which Arnold supports).

The standard libtiff library is all that's necessary to read tiled+mipmapped TIFF, through the use of the appropriate TIFF tags and library calls.

What makes the difference and the main reason to introduce this step into the pipeline is the tiled mipmapped images are much more efficient and cache friendly to the image library OIIO.

### **How are the settings for textures related? Why .tx files?**

▼ [Click here to expand...](#)

The texture cache - recommended settings.

- Recommended settings for maketx and which ones are important.

- Windows batch scripts: If you copy one of these scripts to a .bat file (simple text file with .bat as file extension) in the same folder as maketx.exe and drag a link to it to your desktop, you can throw multiple texture images on it at once and they will get converted in one go. If you don't want the verbose output, remove the "-v", but it may help you understand what maketx does. If you don't want the window

to stay visible at the end, remove the "pause".

Default settings (mipmap and resizing):

```
@for %%i in (*.*) do maketx.exe -v %%i  
pause
```

No mipmapping, but automatic resizing:

```
@for %%i in (*.*) do maketx.exe -v --nomipmap %%i  
pause
```

No resizing and no mipmapping:

```
@for %%i in (*.*) do maketx.exe -v --nomipmap --noresize %%i  
pause
```

### **How do I adjust the falloff of an Arnold light?**

▼ [Click here to expand...](#)

You must attach a "filter" to the light. There are several filters for different needs (e.g. gobo,barndoorsetc). To get a normal decay behavior, you don't need to do anything, since the light node will default to real-world inverse square (i.e., quadratic) decay. If you want further control you will want to use the light decay filter, which provides control over the attenuation ranges.

### **Why does the standard shader compute reflections twice?**

▼ [Click here to expand...](#)

The (deprecated) standard shader can compute both sharp and glossy specular reflections.

Sharp reflections are controlled by the Kr and Kr\_color parameters. This type of reflection uses one ray per shader evaluation, with a depth limited by the GI\_reflection\_depth global option.

Glossy reflections are controlled by the Ks, Ks\_color, specular\_brdf, specular\_roughness and specular\_anisotropy parameters. This type of reflection uses GI\_specular\_samples to determine how many rays to trace per shader evaluation, and the depth is limited by the GI\_specular\_depth global option. Note that using a specular\_roughness of 0 will also give you a sharp reflection, but doing this is slower than using the pure mirror reflection code.

These two types of reflection can coexist and have independent Fresnel controls. They are simply added together with the other components of the standard shader, without taking into account any energy conservation. We will probably end up unifying both types of reflections in the future.

### **What is the re-parameterization of specular\_roughness in the standard surface shader, and why is it non-linear?**

▼ [Click here to expand...](#)

Following the advice of testing from artists, it was found they didn't like the linear mapping of the radius of the specular highlight. They instead preferred a mapping that is slightly curved, with an equation of type  $1/r^3$ . This  $1/r^3$  mapping resulted difficult in the Cook-Torrance and Ward-Duer BRDF cases (requiring expensive powf() calls), and caused the specular\_roughness to lose some of the "physical" sense of being proportional to the specular highlight's radius. We, therefore, opted to square the current roughness parameter. This results in a similar result to the  $1/r^3$  while maintaining some of the "physical" sense of the parameter (instead of doubling the radius each time you double the roughness, you end up quadrupling the radius).

### **How do you capture a 360-degree snapshot from a scene in Latitude-Longitude panoramic format?**

▼ [Click here to expand...](#)

Lat/long maps can be rendered with Arnold's cylindrical camera. There is an example in the SItA trac#638. For more information, from the command-line, type kick -info cyl\_camera. Make sure that the horizontalfov is set to 360, the verticalfov is set to 180, and the camera aspect ratio is set to 1.

### **How does bucket size affect performance?**

▼ [Click here to expand...](#)

To simplify filtering, Arnold adds a bit of "edge slack" to each bucket in each dimension. The amount of "edge slack" is exactly  $2 * \text{pixel filter width}$  (unless you are using a box-1 filter, which has no slack at all). If the user sets the bucket size to 32x32 with a filter width of 2, the buckets are internally extended to 36x36. This is so that each bucket has enough samples in the edges to perform pixel filtering independently of each other bucket, as inter-bucket communication would greatly complicate multithreading.

Here is an example showing the number of camera rays as the filter width is increased:

- 1024x778, AA samples = 1, filter width = 1, traces a total of 796672 camera rays
- 1024x778, AA samples = 1, filter width = 2, traces a total of 900864 camera rays

The corollary is that you should not use buckets that are too small, as the percentage of "redundant" pixels grows inversely proportional to bucket size. The default 64x64 is a good base setting, but 128x128 should be slightly faster.

### **Do any of the light parameters support shader networks?**

▼ [Click here to expand...](#)

Shader networks are only supported in the color parameter of the quad\_light and skydome\_light nodes, and in the light filters (the filters parameter) of all the lights.

### **How does the exposure parameter work in the lights?**

▼ [Click here to expand...](#)

In Arnold, the total intensity of the light is computed with the following formula: color \* intensity \* 2exposure. You can get exactly the same output by modifying either the intensity or the exposure. For example, intensity=1, exposure=4 is the same as intensity=16, exposure=0. Increasing the exposure by 1 results in double the amount of light.

The reasoning behind this apparent redundancy is that, for some people, f-stops are a much more intuitive way of describing light brightness than raw intensity values, especially when you're directly matching values to a plate. You may be asked by the director of photography - who is used to working with camera f-stop values - to increase or decrease a certain light by "one stop". Other than that, this light parameter has nothing to do with a real camera's f-stop control. Also, working with exposure means you won't have to type in huge values like 10.000 in the intensity input if your lights have quadratic falloff (which they should).

If you are not used to working with exposure in the lights, you can simply leave the exposure parameter at its default value of 0 (since  $20 = 1$ , the formula then simplifies to: color \* intensity \* 1).

### **How do the various subdiv\_adaptive\_metric modes work?**

▼ [Click here to expand...](#)

- edge\_length: patches will be subdivided until the max length of the edge is below subdiv\_pixel\_error (regardless of curvature).
- flatness: patches will be subdivided until the distance to the limit surface is below subdiv\_pixel\_error (regardless of patch/edge size). This usually generates fewer polygons than edge\_length and is therefore recommended.
- auto: uses the flatness mode unless there is a displacement shader attached to the mesh, in which case it uses the edge\_length mode. The rationale here is that if you are going to displace the mesh you probably don't want the subdiv engine to leave big polygons in flat areas that will then miss the displacement (which happens at post-subdivided vertices).

### **Should I use a background sky shader or a skydome\_light?**

▼ [Click here to expand...](#)

The skydome\_light should always be preferred, as it supports all functionality of the background while reducing noise. The background shader is considered deprecated. There are various reasons why using skydome\_light is more efficient:

- The skydome\_light uses importance sampling to fire rays to bright spots in the environment, therefore automatically achieving both soft and sharp shadows; sampling the sky shader with GI rays cannot achieve hard shadows in reasonable times, you will need huge amounts of GI samples.
- The environment map lookups for the skydome\_light are cached rather than evaluated at render time. Since texture lookups via OIIO are very slow, this caching results in a nice speedup, usually 2-3x faster than uncached (if you are curious, you can switch to uncached lookups by setting options.enable\_fast\_importance\_tables = false and measure the difference yourself).
- The skydome\_light is sampled with shadow rays, which can be faster than GI rays because shadow rays only need to know that any hit blocks the light (rather than the first hit). This also means the sampling quality for the skydome\_light is controlled via skydome\_light.samples, whereas the quality of a background sky is controlled via the GI\_{diffuse|specular}\_samples. This subtle distinction is very important: skydome\_light is direct lighting and sampled with shadow rays, whereas the background sky shader is indirect lighting and therefore sampled with GI rays.

### **How does the ignore\_list parameter from the options node work?**

▼ [Click here to expand...](#)

It tells the renderer to ignore nodes filtered by type. The following example will ignore, at scene creation, all of the occurrences of Lambert, standard and curves nodes:

```
options
{
  ...
  ignore_list 3 1 STRING lambert standard curves
```

}

### **Which coordinate space does Arnold assume for a vector displacement?**

▼ [Click here to expand...](#)

The displacement shader should output a vector in object space.

### **How do the different subdiv\_uv\_smoothing modes work?**

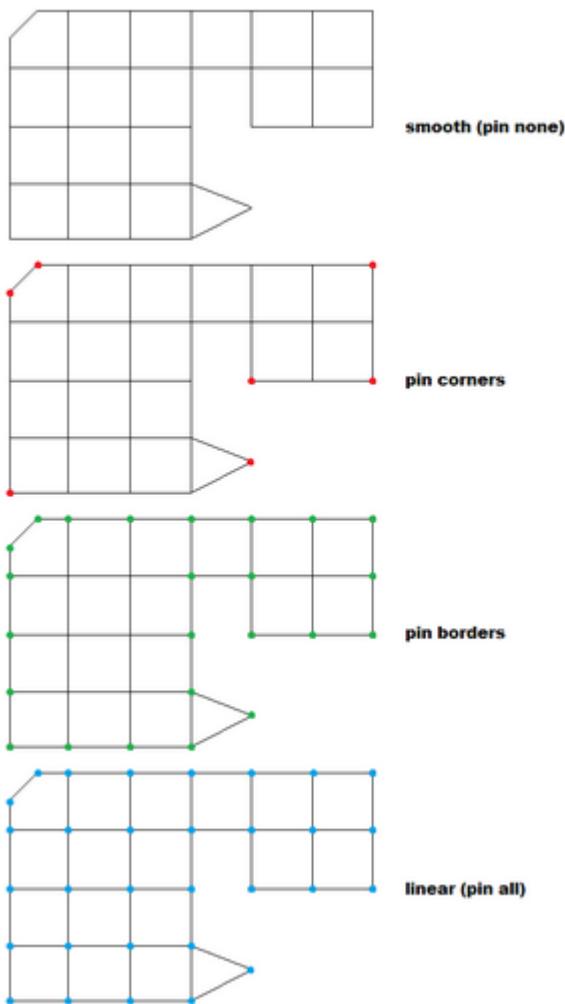
▼ [Click here to expand...](#)

The `subdiv_uv_smoothing` setting is used to decide which sub-set of UV vertices on the control mesh get the Catmull-Clark smoothing algorithm applied to them. Those vertices that do not get smoothing applied to them are considered to be "pinned" since the apparent effect is that their UV coordinates are exactly the same as the corresponding coordinates on the control mesh. The `subdiv_uv_smoothing` modes work as follows:

- `smooth` (none pinned): In this mode Catmull-Clark smoothing is applied to all vertices of the UV mesh, indiscriminately. This mode is really only useful for organic shapes with hidden UV seams whose coordinates do not have to precisely follow any particular geometric edges.
- `pin_corners`: Catmull-Clark smoothing is applied to all vertices of the UV mesh except for those that are connected to only two edges in UV coordinate space (valence = 2). This mode is the default in Arnold (for legacy reasons), however, `pin_borders` is probably a more useful default setting in practice.
- `pin_borders`: Catmull-Clark smoothing is applied to all vertices of the UV mesh except for those that are connected to an edge that forms part of only one polygon. This mode is possibly the most useful of the four and we would suggest trying this one first. In this mode it is guaranteed that the UV coordinate seams on the subdivided mesh will exactly match the corresponding edges of the control mesh, making it much easier to place textures at these seams while still applying Catmull-Clark smoothing on all interior vertices.
- `linear` (all pinned): Catmull-Clark smoothing is not applied to any vertex. This mode can prove useful when rendering objects with textures generated directly on a subdivided surface, like ZBrush models exported in "linear" mode.

The different modes are illustrated in the following image:

#### effects of subdiv\_uv\_smoothing setting on UV coordinates



#### How are the polymesh normals computed when subdivision is enabled?

▼ Click here to expand...

Vertex normals will be computed using the limit subdivision surface, overriding any explicit values set in the nlistparameter. The vertex normals specified in nlist are only used for non-subdivided meshes.

#### What is the Z value for rays that do not hit any object?

▼ Click here to expand...

The Z value for rays that do not hit any object is controlled by the far\_clip parameter on the camera. By default, this camera parameter has a value of 1.0e30f (AI\_INFINITE).

#### Why does iterating through an AiSampler after the first split in the ray tree always return one sample?

▼ Click here to expand...

Splits are typically caused by BSDF integration, which can fire multiple rays for a single AA sample. If this was done at every depth, there would be a combinatorial explosion in the number of rays as the ray depths were increased. Therefore, Arnold "splits" the integrals into multiple rays only once, at the first hit. After that first split, we only follow one single ray, in the spirit of pure path tracing. This may seem like it introduces severe noise, as intuitively the higher bounces seem undersampled compared to the first bounce. But in fact what we are doing is concentrating most of the rays where they matter the most.

Since the AiSampler API was added for users writing their own custom BRDFs, we decided it was best to automatically handle splitting versus no splitting at the renderer level. Thus the sample count is automatically reduced after the first split, which affects the sample counts for both our internal integrators as well as any custom integrators that use AiSampler.

#### How do the motion\_start and motion\_end parameters from the cameras work?

▼ [Click here to expand...](#)

These parameters are used to remap Arnold's "absolute time" to the "relative time" of motion keys, which are specified as values.

`motion_start` and `motion_end` are the times at which the first and last motion key of the shape are sampled. Other motion keys must be uniformly spaced within this range. By convention, the times are frame relative. For example, start and end times -0.5 to 0.5 indicate that the motion keys were sampled midway between the previous and current frame, and current frame and next frame. This is applied to cameras, lights, and shapes.

***Is there a way to iterate over regular and user params for a given AtNode?***

▼ [Click here to expand...](#)

No. Conceptually, they are different things. The parameter list iterator is handled by the `AtNodeEntry`, which defines the internal structure for all nodes belonging to that class. It will traverse all parameter definitions, returning elements of the `AtParamEntry` type, which provides parameter name, type, default value... So, you are traversing the list of parameter definitions, which is stored in the `AtNodeEntry` (i.e. once for all nodes of the same type).

In the case of user data, they only exist in a specific node, and they are not shared by all nodes of the same class. So, for example, you could have some userdata in a poly mesh, and some other differences in another poly mesh, and a third poly mesh with no userdata.

# API Release Notes

- 5.2.0.0
- 5.1.1.2
- 5.1.1.1
- 5.1.1.0
- 5.1.0.1
- 5.1.0.0
- 5.0.2.4
- 5.0.2.3
- 5.0.2.2
- 5.0.2.1
- 5.0.2.0
- 5.0.1.5
- 5.0.1.4
- 5.0.1.3
- 5.0.1.2
- 5.0.1.1
- 5.0.1.0
- 5.0.0.3
- 5.0.0.2
- 5.0.0.1
- 5.0.0.0
- 4.2.9.0
- 4.2.8.0
- 4.2.7.0
- 4.2.6.0
- 4.2.4.0
- 4.2.3.0
- 4.2.2.0
- 4.2.16.0
- 4.2.15.0
- 4.2.14.0
- 4.2.13.0
- 4.2.12.0
- 4.2.11.0
- 4.2.10.0
- 4.2.1.2
- 4.2.1.1
- 4.2.1.0
- 4.2.0.6
- 4.2.0.0
- 4.1.3.5
- 4.1.3.3
- 4.1.3.2
- 4.1.3.1
- 4.1.3.0
- 4.1.2.0
- 4.1.1.0
- 4.1.0.0
- 4.0.9.1
- 4.0.9.0
- 4.0.8.0
- 4.0.7.0
- 4.0.6.0
- 4.0.5.3
- 4.0.5.2
- 4.0.5.1
- 4.0.5.0
- 4.0.4.0
- 4.0.3.0
- 4.0.2.0
- 4.0.16.2
- 4.0.16.1
- 4.0.16.0
- 4.0.15.1
- 4.0.15.0
- 4.0.14.0

- 4.0.13.1
- 4.0.13.0
- 4.0.12.0
- 4.0.11.0
- 4.0.10.2
- 4.0.10.1
- 4.0.10.0
- 4.0.1.3
- 4.0.1.2
- 4.0.1.1
- 4.0.1.0
- 4.0.0.0
- 3.x

## 5.2.0.0

### 5.2.0.0

#### Enhancements

- **Texture baking:** A new type of node called `uv_camera` has been added that will produce an image of a given polymesh's shaded UV space as output, which can be useful for texture baking. (#6091, #7206)
- **Improved sampling of spherical lights:** A new technique for sampling point lights has been added which can show significant reductions in noise, especially for large lights illuminating surfaces at grazing angles (rim lighting, for example). (#5534)
- **Faster adaptive subdivision:** Adaptive subdivision is now up to 2x to 3x faster even on a single thread. In addition, the adaptive codepath has been multi-threaded to fully take advantage of machines with many cores. The aggregated speedup in such machines can be 15x or more. (#2311, #7186, #7201, #7229)
- **Improved EXR read performance:** Threaded read performance and scaling of OpenEXR files has been greatly improved. (#6605)
- **noise denoiser improvements:** The stability and usability of the high-quality `noise` denoiser has been improved thanks to various bugfixes and improved error checking. In particular, the original metadata, display windows, bitdepth and compression are preserved in output files. (#7226)
- **OptiX denoiser improvements:** The GPU memory consumption of the fast OptiX denoiser has been greatly reduced proportionally to the number of denoised AOVs. Fringing artifacts around HDR pixels have been reduced. (#6885, #7100, #7190, #7333, #6880)
- **Sheen in standard\_surface:** The `standard_surface` shader supports a new, energy-preserving sheen effect designed to render cloth-like microfiber materials such as velvet. The sheen effect is layered on top of the diffuse and subsurface components. (#7234)
- **New cell\_noise shader:** A new `cell_noise` shader has been added which can create many different useful cell-like patterns. The color of each cell is mapped to a palette parameter, enabling the easy creation of patterns with colors chosen from a specific palette. (#5985, #6051)
- **New controls in range shader:** The range shader has been augmented with parameters to control contrast, bias and gain. (#7277)
- **RGB clamping in clamp shader:** The clamp shader can now be configured to either a scalar or color mode. (#7278)
- **Matrix shaders:** The `matrix_multiply_vector` and `matrix_transform` shaders have been reinstated. (#7243)
- **Built-in Cryptomatte:** Cryptomatte AOV shaders and filters are now being included as a part of the Arnold core package. (#7301)
- **New built-in volume AOVs:** The Z depth for the first volume contribution can now be output in a flat AOV with `volume_z` (depth AOV for volumes was already available in deep files). Also, `ID` now works for volumes. (#7326, #7327)
- **New control in toon shader:** Edge detection can now be controlled using a STRING type user data called `toon_id`. This feature is enabled when `user_id` is checked. Otherwise, the detected edges will be driven by the object's own name as a toon-specific ID. (#7125)
- **Alembic procedural improvements:** The Alembic library has been updated to 1.7.5 in this release. User data parameters that clash with shape parameters will now get an underscore prefix instead of a warning. Added an `object_transform` parameter to allow additional transformations on the generated geometry. Added a `make_instance` parameter so that the Alembic procedural will automatically create instances of objects present in multiple Alembic procedurals (experimental; disabled by default) (#6916, #6947, #7076, #7109, #7163, #7242, #7261, #7286)
- **Improved operator assignments:** Assignment expressions in operators have improved functionality with regards to reference and string types. (#7284, #7287)
- **Upgrade to OSL 1.9.9:** This upgraded version of OSL addresses several reported limitations involving locales, the availability of certain noise types, and compatibility issues with utility functions like `transformc` being promoted to built-in function definitions. (#6225)
- **Updated to RLM 12.4BL2:** The RLM license server and library have been upgraded from version 12.2BL2 to 12.4BL2, which fixes sporadic access violations and hangs. (#7350, #7120)

#### API additions

- **Sheen closure:** The `AiSheenBSDF()` function has been added which provides the sheen closure used by `standard_surface` so that the same effect can be easily obtained in custom shaders. This function also returns a weight that can be used to layer the sheen closure onto other closures in an energy conserving fashion. (#7234)
- **Multiple Universes:** A new concept has been added to Arnold's API which gives it the capability of creating nodes in any number of user-defined workspaces called `AtUniverse` that can be created and destroyed via the `AiUniverse()` and `AiUniverseDestroy()` functions, respectively. Upon creation, nodes are assigned a universe which will take on their exclusive ownership, and this ownership can be inspected via the `AiNodeGetUniverse()` function. Please note that the "null" universe is the "default" universe, which for the time being is the only one which permits rendering and procedural expansion. (#4129)

#### Incompatible changes

- **Code compatibility:** Code made for Arnold 5.1 should still function in Arnold 5.2 after a recompilation.
- **Binary compatibility:** Even though 5.2 is technically an API-breaking release, Arnold 5.0 and 5.1 plugins (shaders, filters, etc) in the majority of cases will still be compatible with 5.2 and can continue to be used without being recompiled, while procedural plugins on

the other hand will in the majority of cases require recompilation. (#6822)

- **Multiple Universes:** In order to support multiple universes, several existing API functions have a new `AtUniverse` pointer as first parameter indicating which universe these functions must operate on. This change to function signatures implies a break in binary compatibility with any client code compiled for previous versions of Arnold that were using these functions. However a set of function overloads that preserve the previous signatures has been added to the C++ headers such that the mismatched signatures can be resolved through a code recompilation. The API functions affected by this change are:

- `AiASSWrite()`
- `AiASSWriteWithMetadata()`
- `AiASSLoad()`
- `AiNode()`
- `AiNodeLookUpByName()`
- `AiUniverseCacheFlush()`
- `AiUniverseGetOptions()`
- `AiUniverseGetCamera()`
- `AiUniverseGetSceneBounds()`
- `AiUniverseGetNodeIterator()`

- **Removed texture blur options :** The `texture_specular_blur`, `texture_diffuse_blur`, and `texture_sss_blur` options have been removed after having defaulted at 0 for some time. They are no longer needed for improving texture performance since Arnold is able to automatically blur the textures as needed without the over-blurring that these options would produce. (#4706)
- **GPU denoising options renamed:** The GPU-related render options used by the OptiX denoiser have been renamed to use the `gpu_*` prefix. (#7190)
- **ENUM params now byte-sized:** Enum parameters are now internally stored as bytes instead of ints, and will be reported as such by the `AiParamGetTypeSize()` API. (#7114)

## Bug fixes

- #7325 procedural loading a .ass containing many nodes that reference other nodes is very slow
- #6472 OSL issues with Locale
- #6916 curves in Alembic proc cut short
- #6947 Skip invalid curves
- #7012 Noise: black pixels around variance spikes
- #7027 Non-ascii characters prevent Open EXR images from being loaded
- #7076 Rename user data parameters that clash with shape nodes in alembic expansion
- #7154 Adaptive Subdivision: crash with linear patches and different position and UV topologies
- #7179 Adaptive Subdiv: irregular patch crash with different position and uv topologies
- #7226 Noise: support inputs with different data windows
- #7286 Alembic object transform
- #7303 Python binding fixes for functions that return `AtNode` types
- #7312 Raw drivers crash (including Cryptomatte's manifest driver)
- #7341 Crash when computing the render stats for procedural
- #7355 Alembic curve user data expansion

## 5.1.1.2

### 5.1.1.2

#### Bug fixes

- #7259 linear normal subdivision: incorrect normals with multiple keys
- #7299 Regex/operators memory leak
- #6370 Procedural cache not working with custom namespaces
- #7230 Object “id” parameter overridden by procedural
- #7242 Fix alembic array attribute expansion for regular properties
- #7244 Crash when loading empty points through a procedural
- #7261 Alembic curves sets radius on curves and points to width
- #7267 linking to only some of the mesh or quad light’s color components gives incorrect results
- #7289 Allow reading files marked with a UTF-8 encoding
- #7306 AiShaderGlobals() can cause crashes when called from multiple threads

### 5.1.1.1

#### Bug fixes

- #6771 Incomplete render when you restart after interrupting a render during displacement/subdivision
- #7180 noice crashes with separate float Z input file
- #7213 MaterialX: Displacements and mixing of generic/texture shaders in materials now supported using a shader type context
- #6926 MaterialX: Resolve environment variables in search paths
- #6993 noice should preserve metadata, display windows, and add noice args and version metadata
- #6994 Noice should preserve channel bit width and compression for outputs
- #7094 Procedural containing ginstances are evaluated before instanced node
- #7113 Contour lines shouldn’t be drawn when width\_scale is zero
- #7124 Memory leak when interrupting displacement
- #7127 Procedural ginstance matrices are accumulated at each render
- #7130 Render incorrectly aborted in applyDevice when GPU is not needed
- #7152 noice should exit when unsupported formats are used
- #7163 Array property fixes in alembic procedural
- #7176 Motion blur on instances of procedurals
- #7187 Fix op target selection in kick command line
- #7191 Slight error in checkerboard shader
- #7193 noice broken sequence handling when temporal denoising is not used
- #7198 Alembic procedural uses inheritsXforms property
- #7200 Output metadata: output correct metadata for single layer files

### 5.1.1.0

#### Enhancements

- **Improved stability with incompatible OptiX versions:** We have removed the hard dependency on the Nvidia OptiX library needed for GPU denoising. We can now prevent DLL conflicts when users have installed renderers from other vendors that link to a different/older version of the OptiX libraries, which was causing plugins (such as MtoA) to refuse to load. (#6959)
- **More efficient texture mapping:** EXR textures can now be read more quickly. In addition, the texture\_diffuse\_blur option is now defaulted to 0 so that sharper and more accurate diffuse reflections can be seen without texture performance being negatively impacted. This option is now deprecated and, if no problems are raised, in a future release we will remove this and the other texture\_\*\_blur options. (#4692, #6605, #6890)
- **noice sequence handling and extra frames:** A new argument (-f n or --frames n) has been added to noice to denoise image sequences. Another new argument (-ef n or --extraframes n) specifies how many additional source frames before and after the current one should be used, for improved stability in animation sequences. (#6899) In this example, noice will run for 10 consecutive frames starting at frame 5, taking into account two frames before and two frames after each source frame (e.g. source frame number 5 will take into account frames 3, 4, 5, 6, and 7):

```
noice -i mysequence.0005.exr -o denoised.0005.exr -f 10 -ef 2
```

- **noice support for AOVs from multiple files:** noice now supports multiple input files to be denoised. These can be specified by using the -i (--input) argument as many times as needed. Multiple output files are also supported to preserve the original AOV layout. Denoised output files matching the input files can be specified with the -o (--output) argument as many times as needed. This is compatible with sequence arguments (-f and -ef) and with manually specified extra frames. As always, files from the frame to be denoised should come first. (#6826) In the following example, the input feature AOVs and the input AOVs to be denoised were rendered in separate files, and denoised results will be saved in two separate files to preserve the original AOV layout:

```
noice -i RGBA.exr -i light_AOVs.exr -i features.exr -l light_AOV1 -l  
light_AOV2 -o denoised_RGBA.exr -o denoised_light_AOVs.exr
```

- **noice AOV denoising:** AOVs are now fully denoised individually for increased quality at the expense of run time. Also, per-AOV variances will be used if present. Every RGB channel to be denoised will benefit from an associated variance AOV. Additionally, RGBA is no longer required to denoise AOVs from a given EXR file. (#7011, #7015)
- **noice threads:** a new command line flag (-t or --threads) has been added to specify the number of threads to use. This is only available on Windows and Linux. The flag works like the similar kick -t <integer>: a negative number will leave some cores unused (-1 will leave one core unused), a positive number will use the specified number of cores, and 0 will use all cores. (#7017)
- **noice custom features:** for advanced users, the new flag (-fe or --features) specifies which AOVs will be used as features to guide denoising. For instance -fe volume\_albedo will use volume\_albedo to denoise instead of the default diffuse\_albedo. (#7070)
- **More robust Alembic procedural:** We have improved the robustness of the built-in Alembic procedural with various bug fixes, including support for facevarying user data, and support for array properties. (#6909, #6937, #7025)
- **Support for Microsoft Azure Sovereign Clouds:** In addition to regular Azure cloud, Arnold batch rendering is now entitled in the Microsoft National Clouds. (#6960)

## API additions

- AiProfileGetFileName(): To match AiProfileSetFileName(), we now also have the corresponding AiProfileGetFileName(). (#7031)

## Incompatible changes

- materialx.mtlx renamed: The mtlx parameter of the materialx operator node, which should point to a .mtlx file on disk, has been renamed to filename. The old name mtlx is kept as a deprecated synonym so that existing scenes don't break. Also, if the file cannot be opened, the materialx node will now report an error (and abort the render) instead of a warning. (#7018)

## Bug fixes

- #6844 Catch exceptions in MaterialX operator to not crash with invalid documents
- #6903 Incorrect cooker cache check prevents some upstream operators from cooking
- #6988 Noise: non ASCII characters in non-existing path cause crash on Windows
- #6999 Named MaterialX node graph outputs not correctly bound to shader inputs
- #6596 `aov\_write\_float` shader not respecting `standard\_surface.transmit\_aov` flag
- #6735 Enabling transmission affects rayDepth AOV
- #6849 Cannot render rgb masks for transmissive objects
- #6896 Object-To-World transform Uninitialized in OSL
- #6900 Noise should ignore precision errors in depth or normal buffers

- #6902 Procedural load crashing due to invalid nodes
- #6909 Match number of motion keys for normals and vertex list in alembic procedural
- #6913 Fix crash when optix denoiser DLLs are not found
- #6923 Crash when loading empty polymesh through a procedural
- #6924 Optix denoiser crashes after changing device because buffers still mapped
- #6928 Crash caused by old versions of NVML
- #6937 Add facevarying userdata support to alembic procedural
- #6938 crash due to an invalid/empty procedural within another procedural
- #6944 Crash in consecutive AiBegin sessions on machines with unsupported Nvidia drivers
- #6948 Evaluation of uninitialized debug closure when running aov shaders
- #6951 Crash when using AiNodeClone in procedurals with parallel\_node\_init
- #6952 Noise: +inf depth values cause issues in background
- #6970 Hang due to ginstances referencing a non-existing node
- #6977 standard\_hair crash with specular tints set to 0
- #7006 kick -nokeypress doesn't work
- #7025 Array properties in alembic procedural not translated
- #7038 Typo in JSON shader stats
- #7061 Failure to write JSON stats when NaN or Inf encountered
- #7062 Using toon shader as an aov shader crashes when a background shader exists
- #7078 Noise: pixel NaNs when using parallel denoising with black RGBA
- #7083 warn when several drivers write to the same file
- #6942 AOV output string error when layer name matches scene node name

## 5.1.1.1

### 5.1.1.1

#### Bug fixes

- #6771 Incomplete render when you restart after interrupting a render during displacement/subdivision
- #7180 noice crashes with separate float Z input file
- #7213 MaterialX: Displacements and mixing of generic/texture shaders in materials now supported using a shader type context
- #6926 MaterialX: Resolve environment variables in search paths
- #6993 noice should preserve metadata, display windows, and add noice args and version metadata
- #6994 Noice should preserve channel bit width and compression for outputs
- #7094 Procedural containing ginstances are evaluated before instanced node
- #7113 Contour lines shouldn't be drawn when width\_scale is zero
- #7124 Memory leak when interrupting displacement
- #7127 Procedural ginstance matrices are accumulated at each render
- #7130 Render incorrectly aborted in applyDevice when GPU is not needed
- #7152 noice should exit when unsupported formats are used
- #7163 Array property fixes in alembic procedural
- #7176 Motion blur on instances of procedurals
- #7187 Fix op target selection in kick command line
- #7191 Slight error in checkerboard shader
- #7193 noice broken sequence handling when temporal denoising is not used
- #7198 Alembic procedural uses inheritsXforms property
- #7200 Output metadata: output correct metadata for single layer files

## 5.1.1.0

### 5.1.1.0

#### Enhancements

- **Improved stability with incompatible OptiX versions:** We have removed the hard dependency on the Nvidia OptiX library needed for GPU denoising. We can now prevent DLL conflicts when users have installed renderers from other vendors that link to a different/older version of the OptiX libraries, which was causing plugins (such as MtoA) to refuse to load. (#6959)
- **More efficient texture mapping:** EXR textures can now be read more quickly. In addition, the `texture_diffuse.blur` option is now defaulted to 0 so that sharper and more accurate diffuse reflections can be seen without texture performance being negatively impacted. This option is now deprecated and, if no problems are raised, in a future release we will remove this and the other `texture_*_blur` options. (#4692, #6605, #6890)
- **noice sequence handling and extra frames:** A new argument (`-f n` or `--frames n`) has been added to `noice` to denoise image sequences. Another new argument (`-ef n` or `--extraframes n`) specifies how many additional source frames before and after the current one should be used, for improved stability in animation sequences. (#6899) In this example, `noice` will run for 10 consecutive frames starting at frame 5, taking into account two frames before and two frames after each source frame (e.g. source frame number 5 will take into account frames 3, 4, 5, 6, and 7):

```
noice -i mysequence.0005.exr -o denoised.0005.exr -f 10 -ef 2
```

- **noice support for AOVs from multiple files:** `noice` now supports multiple input files to be denoised. These can be specified by using the `-i` (`--input`) argument as many times as needed. Multiple output files are also supported to preserve the original AOV layout. Denoised output files matching the input files can be specified with the `-o` (`--output`) argument as many times as needed. This is compatible with sequence arguments (`-f` and `-ef`) and with manually specified extra frames. As always, files from the frame to be denoised should come first. (#6826) In the following example, the input feature AOVs and the input AOVs to be denoised were rendered in separate files, and denoised results will be saved in two separate files to preserve the original AOV layout:

```
noice -i RGBA.exr -i light_AOVs.exr -i features.exr -l light_AOV1 -l  
light_AOV2 -o denoised_RGBA.exr -o denoised_light_AOVs.exr
```

- **noice AOV denoising:** AOVs are now fully denoised individually for increased quality at the expense of run time. Also, per-AOV variances will be used if present. Every RGB channel to be denoised will benefit from an associated variance AOV. Additionally, `RGBA` is no longer required to denoise AOVs from a given EXR file. (#7011, #7015)
- **noice threads:** a new command line flag (`-t` or `--threads`) has been added to specify the number of threads to use. This is only available on Windows and Linux. The flag works like the similar `kick -t <integer>`: a negative number will leave some cores unused (-1 will leave one core unused), a positive number will use the specified number of cores, and 0 will use all cores. (#7017)
- **noice custom features:** for advanced users, the new flag (`-fe` or `--features`) specifies which AOVs will be used as features to guide denoising. For instance `-fe volume_albedo` will use `volume_albedo` to denoise instead of the default `diffuse_albedo`. (#7070)
- **More robust Alembic procedural:** We have improved the robustness of the built-in Alembic procedural with various bug fixes, including support for facevarying user data, and support for array properties. (#6909, #6937, #7025)
- **Support for Microsoft Azure Sovereign Clouds:** In addition to regular Azure cloud, Arnold batch rendering is now entitled in the Microsoft National Clouds. (#6960)

#### API additions

- `AiProfileGetFileName()`: To match `AiProfileSetFileName()`, we now also have the corresponding `AiProfileGetFileName()`. (#7031)

#### Incompatible changes

- `materialx.mtlx` renamed: The `mtlx` parameter of the `materialx` operator node, which should point to a `.mtlx` file on disk, has been

renamed to filename. The old name mtlx is kept as a deprecated synonym so that existing scenes don't break. Also, if the file cannot be opened, the materialx node will now report an error (and abort the render) instead of a warning. (#7018)

## Bug fixes

- #6844 Catch exceptions in MaterialX operator to not crash with invalid documents
- #6903 Incorrect cooker cache check prevents some upstream operators from cooking
- #6988 Noise: non ASCII characters in non-existing path cause crash on Windows
- #6999 Named MaterialX node graph outputs not correctly bound to shader inputs
- #6596 `aov\_write\_float` shader not respecting `standard\_surface.transmit\_aov` flag
- #6735 Enabling transmission affects rayDepth AOV
- #6849 Cannot render rgb masks for transmissive objects
- #6896 Object-To-World transform Uninitialized in OSL
- #6900 Noise should ignore precision errors in depth or normal buffers
- #6902 Procedural load crashing due to invalid nodes
- #6909 Match number of motion keys for normals and vertex list in alembic procedural
- #6913 Fix crash when optix denoiser DLLs are not found
- #6923 Crash when loading empty polymesh through a procedural
- #6924 Optix denoiser crashes after changing device because buffers still mapped
- #6928 Crash caused by old versions of NVML
- #6937 Add facevarying userdata support to alembic procedural
- #6938 crash due to an invalid/empty procedural within another procedural
- #6944 Crash in consecutive AiBegin sessions on machines with unsupported Nvidia drivers
- #6948 Evaluation of uninitialized debug closure when running aov shaders
- #6951 Crash when using AiNodeClone in procedurals with parallel\_node\_init
- #6952 Noise: +inf depth values cause issues in background
- #6970 Hang due to ginstances referencing a non-existing node
- #6977 standard\_hair crash with specular tints set to 0
- #7006 kick -nokeypress doesn't work
- #7025 Array properties in alembic procedural not translated
- #7038 Typo in JSON shader stats
- #7061 Failure to write JSON stats when NaN or Inf encountered
- #7062 Using toon shader as an aov shader crashes when a background shader exists
- #7078 Noise: pixel NaNs when using parallel denoising with black RGBA
- #7083 warn when several drivers write to the same file
- #6942 AOV output string error when layer name matches scene node name

## 5.1.0.1

### 5.1.0.1

#### Bug Fixes

- #6844 Catch exceptions in MaterialX operator to not crash with invalid documents
- #6903 Incorrect cooker cache check prevents some upstream operators from cooking
- #6896 Object-To-World transform Uninitialized in OSL
- #6900 Noise should ignore precision errors in depth or normal buffers
- #6902 Procedural load crashing due to invalid nodes
- #6913 Fix crash when optix denoiser DLLs are not found
- #6923 Crash when loading empty polymesh through a procedural
- #6924 Optix denoiser crashes after changing device because buffers still mapped
- #6928 Crash caused by old versions of NVML
- #6938 crash due to an invalid/empty procedural within another procedural

## 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_k_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_samping` 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 (noise):** a stand-alone, post-process denoiser executable called `noise` 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`

`re_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_paramete`r, `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_u_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

## 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_k_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_samping` 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 (noise):** a stand-alone, post-process denoiser executable called `noise` 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_u_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

## 5.0.2.4

### 5.0.2.4

- #6624 Memory leak when destroying custom procedurals
- #6537 standard\_hair no longer creates the 'hair' LPE label
- #6554 Report virtual memory when allocation fails
- #6568 OSL shader param cannot link to alpha component of RGBA shader.
- #6582 Color Manager: fix python binding for AiColorManagerTransform return value
- #6585 allow IOR below 1.0
- #6589 camera in camera\_projection should be linkable
- #6600 some curves are incorrectly clipped
- #6610 Potential crashes with user data of string type
- #6630 Stack overflow during OSL node conversion traversal
- #6646 Number of image tags too low
- #6555 Provide simple constructor for matrix in Python API

### 5.0.2.3

- #6511 UDIMs: incorrect character substitutions
- #6481 crash during accel construction with invalid or invisible geometry
- #6500 error building BVH over procedurals with no visible objects
- #6516 Properly warn when there are more than 15 per-light AOVs
- #6527maketx and AiMakeTx are extremely slow or crash with certain large images

### 5.0.2.2

- #6257 s/twrap not working with <attr> expressions set in the image node filename
- #6411 Excessive memory use with compressed UVs
- #6417 Small PolyMesh memory leak at render shutdown
- #6422 OSL raytype mismatch
- #6440 Handle invalid characters when image tokens are resolved
- #6441 Backfacing normals not supported in normal\_map
- #6442 Chained bumps using bump2d isn't working
- #6453 Custom procedurals cannot create nodes before proc\_init
- #6454 Crash With Linear Subdivision, Vertex Normals, and Deformation Blur

### 5.0.2.1

- #6409 ASS metadata load is too slow
- #6412 Crash when setting an array user parameter with its current value

## 5.0.2.0

### Enhancements

- **New sub-surface scattering algorithm:** A new, more accurate way of calculating SSS has been added. Unlike the current empirical BSSRDF method based on diffusion theory, this new method actually traces below the surface with a real random walk and makes no assumptions about the geometry being locally flat. This means it can take into account anisotropic scattering like brute-force volume rendering and produces much better results around concavities and small details. It can also be substantially faster for large scattering radius (i.e. large mean free path) compared to the old method. On the other hand, the new method can be slower in dense media (i.e. small mfp), does not support `sss_setname` for blending two surfaces together, may require redialing materials to achieve a similar look, and is more sensitive to non-closed meshes, "mouth bags", and internal geometry potentially casting shadows. This new algorithm is exposed in the `standard_surface` shader via the new parameters `subsurface_type` (with enum values `diffusion` and `randomwalk`) and `subsurface_anisotropy` (Henyey-Greenstein's eccentricity `g` from -1.0 to +1.0). The default is to use the old empirical diffusion method in order not to break the look of existing scenes.
- **Car paint shader:** We are now shipping a dedicated shader for car paint, which can be thought of as the combination of a simplified version of the `standard_surface` and `flakes` shaders. This shader can create a wide range of car paint looks without having to connect several nodes. For example, a pearlescent effect can be easily added to both the specular and flakes layers by simply tweaking a few parameters such as `specular_flip_flop` and `flake_flip_flop`. An arbitrary number of layers of flakes can be used (`flake_layers`). The flakes at a deep layer are covered by the ones closer to the surface and more tinted by pigments (specified by the `transmission_color` parameter).
- **Subdivision frustum culling:** Subdivision patches outside the view or dicing camera frustum will not be subdivided. This is useful for any extended surface that is only partially visible as only the directly visible part will be subdivided. Similarly, no subdivision work will happen if a mesh is not directly visible. This can be turned on globally by setting `options.subdiv_frustum_culling true` and can be turned off for specific meshes with `polymesh.subdiv_frustum_ignore true`. The global `options.subdiv_frustum_padding` adds a world space padding to the frustum that can be increased as needed to minimize artifacts from out-of-view objects in cast shadows, reflections, etc. Note that motion blur is not yet taken into account and moving objects might require some additional padding.
- **Improved accuracy of UV coords:** Mesh UV coordinates are now handled with higher numerical precision, fixing jagged artifacts that sometimes appeared when using high-resolution UDIM textures over very wide UV ranges.
- **Improved volume sampling of low-spread lights:** Low-spread quad and disk lights should now produce less noise and show increased performance when participating in atmospheric scattering effects.
- **Improved procedural namespace memory usage:** Reduced memory used by procedural namespacing by about 28KB per procedural primitive, so that procedurals now use very little additional memory. In a scene with 100K procedurals, this gave about a 2.7GB reduction in memory use.
- **Faster opacity masks:** Texture map-based opacity masks will now be faster to render. Testing indicates around 3-13% faster.
- **Faster IPR:** The message logging system has been optimized, resulting in about 10% faster performance in IPR, like while moving the camera. In addition, the IPR mode in `kick` has been made substantially more responsive.
- **Faster AiNodeDestroy:** we have optimized removal time for nodes contained in procedurals, greatly reducing the shutdown time at the end of a render in complex scenes.
- **Faster triplanar shader:** Texture filtering in the `triplanar` shader has been improved, giving better antialiasing and up to a 2x speedup, specially when using high-resolution texture maps.
- **Cellular option in triplanar shader:** The `triplanar` shader now supports projection through Voronoi cells using the new `cell` parameter. The rotation angle of the projected texture for each cell can be controlled with the `cell_rotate` parameter. Cells can be smoothly blended using the `cell_blend` parameter.
- **Improved flakes shader:** The `size` parameter is replaced by the `density` parameter, which makes it easy to control the size and number of flakes. Alpha channel can be used as a mask. The new `flakes` shader supports non-disc shapes and 3D flakes, which are useful to render gemstone inclusions like goldstone, for example.
- **Improved shadow\_matte shader:** We have revamped and simplified the shader to make it easier to use, and fixed a number of long-standing issues: Indirect illumination now fills the global `diffuse_indirect` and `specular_indirect` AOVs, so we have removed the shader's (confusingly named) `indirect_diffuse` and `indirect_specular` AOVs. Self-reflections are no longer rendered. A new `specular_IOR` parameter was added that controls Fresnel reflection. Parameters `offscreen_color` and `background_type` were removed. The new enum parameter `background` can be set to either `scene_background` (default) or `background_color`, which allows to connect a specific texture in the `background_color` parameter slot. Parameter `alpha_mask` was added to control whether the alpha must be opaque or if it has to contain the shadow mask.
- **Support for more OSL attributes:** OSL shaders now support `getattribute()` lookups of standard camera attributes (e.g. `camera:fov`, `camera:resolution`, etc) as well as the geometry attributes `geom:type`, `geom:name`, `geom:bounds`, and `geom:objbounds` on objects.
- **Transmit AOVs and Alpha:** The `standard_surface` shader with transmission can now pass through AOVs, by enabling the `transmit_aovs` parameter. If the background is transparent, then the transmissive surface will become transparent so that it can be composited over another background. Light path expression AOVs will be passed through, so that for example a diffuse surface seen through a transmissive surface will end up in the `diffuse` AOV. Other AOVs can also be passed straight through (without any opacity blending), which can be used for creating masks for example.
- **Improved multi-threaded render time stats:** Render times when using more than one thread did not really work. We have improved this so that render times are now much more reliable and useful and can now be confidently used to determine what parts of Arnold are the most expensive. In particular, the subdivision and displacement times will now show how much total time was used

as well as what fraction of that time was spent with threads unable to do useful work. This "threads blocked" time can often be lowered by using larger buckets or the `random bucket_scanning` ordering.

- **Custom procedural namespaces:** Procedurals can now declare a custom namespace using the new `namespace` parameter. This custom namespace can be used instead of the procedural name, to reference contents through absolute or relative paths. Multiple procedurals can share the same namespace by using the same custom name. Also, they can declare an empty name and they will use the global namespace. (#6085)
- **Added `-turn_smooth` option to `kick`:** When using `kick -turn`, the `-turn_smooth` flag can be added to smoothly start and stop the movement as the original position is reached with a cubic ramp.
- **Added `-laovs` option to `kick`:** Using `kick -laovs file.ass` will display a list of all the built-in AOVs and all the AOVs registered by this `.ass` file.
- **Added `cputime heatmap` view to `kick`:** When using `kick` you can now toggle between viewing kicks default output and a cputime heatmap with the `T` key. The mapping of the heat map can be scaled with the `[` and `]` keys.
- **Support for `wasd` keys in `kick -ipr m`:** Running `kick` in Maya-style IPR mode with `-ipr m` now supports "`wasd`" style keyboard movement. This was previously only available in Quake-style mode, `-ipr q`.
- **`maketx` version info:** The custom `maketx` binary that ships with Arnold now reports, both in the command-line and in the embedded EXR headers, that it was built specifically for "OpenImageIO-Arnold", to distinguish it from the official "OpenImageIO" one.
- **`maketx` color spaces:** The custom `maketx` that ships with Arnold now supports OCIO and SynColor (when available) through the `colorengine`, `colorconfig` and `colorconvert` flags. See `maketx --help`.
- **AiMakeTx reports info messages:** AiMakeTx now also prints out OIIO informational messages (generated in verbose mode, for instance) in addition to errors.
- **AiMakeTx releases input file lock sooner:** AiMakeTx will now close the input texture as soon as possible instead of waiting for all the maketx jobs to finish.
- **AiMakeTx and maketx optimized flags:** Both AiMakeTx and maketx now always run with the flags `--monochrome-detect` `--opaque-detect` `--constant-color-detect` `--fixnan` `box3` `--oio`, which can result in smaller `.tx` files that are faster to load and take less memory.
- **Report when textures are changed during render:** The log files now report when texture modifications during a render cause a texture read error, which can happen in certain pipelines.
- **OCIO color space family support:** The OCIO color manager now implements color space enumeration by family. This is useful for UI drop down organization.
- **OCIO view/display enumeration:** The OCIO color manager can now enumerate view/display combinations using the "View (Display)" family. This lets client programs filter color spaces when only a display transform is appropriate.
- **.ass metadata from compressed files:** We now support loading metadata from `.ass.gz` files through the `AiMetadataStoreLoadFromASS()` API function.
- **Upgraded to OIIO 1.7.17:** OpenImageIO has been upgraded to OIIO 1.7.17.

## API additions

- **Color space family enumeration:** Existing color space families for the current config can be enumerated using the new API methods `AiColorManagerGetNumFamilies` and `AiColorManagerGetFamilyNameByIndex`. The addition of these new API methods requires any existing custom color managers (which we know are very rare) to be recompiled.
- **AiAOVSampleIteratorGetPixel():** Custom filters can now determine what pixel is being filtered with the new API method `AiAOVSampleIteratorGetPixel()`.
- **transparent LPE label:** When setting the `transparent` LPE label on a BSDF, the surface will act as if it is transparent and pass through AOVs. This would typically be used for transmission BSDFs, as it is in the `standard_surface` shader.
- **Deprecated API warnings:** Defining `AI_ENABLE_DEPRECATED_WARNINGS` will cause the compiler to emit warnings if deprecated Arnold API functions are used.
- **Random walk SSS closure:** The new random walk SSS algorithm is exposed in the C++ API as `AiClosureRandomWalkBSSRDF()`. The corresponding OSL closure is `randomwalk_bssrdf`.

## Incompatible changes

- **autotile disabled by default:** We found that the `autotile` code in OIIO does not scale with high-resolution textures. In order to avoid very slow loading of untextured textures (such as JPEG) when `autotile` was enabled, we have now changed the `autotile` default setting to 0, which effectively disables it. In very rare cases, when rendering with a large number of high-resolution untextured textures, this change might degrade performance as the texture cache will blow up. The real solution is to never use untextured files, and instead convert all untextured textures to `.tx` tiled files.
- **maketx dependencies:** The custom `maketx` that ships with Arnold now depends dynamically on `libai.so` and optionally on `sync_olor_shader.so`, therefore to work correctly it needs to be run from its installation folder, like `kick`, or alternatively the new dependencies should be copied to where `maketx` is running from.
- **Light groups and volume shading:** Just like with surface shapes, Arnold now obeys light group assignment on volume shapes and surfaces with volumetric interiors. While in many cases desirable, this can produce an unexpected change in the final image in scenes with light group assignments.
- **flakes shader:** It was not easy to control the number of flakes with the `size` and `scale` parameters because they were mutually dependent. Now this can be easily done using the new `density` parameter. The shape of each flake has been changed from disc to

Voronoi cell, which is more suitable to render inclusions of gem stones. The shader output type has been changed from RGB to RGBA to support a mask.

- **`motionvector` AOV:** The motion vector scaling factor in the built-in `motionvector` AOV has changed. This was required to fix a bug that caused zero motion vectors for certain shutter positions. The output from the `motion_vector` shader is unchanged: it can be used as a workaround in your old scenes if you require the previous scaling.
- **`shadow_matte` changes:** The shader AOVs `indirect_diffuse` and `indirect_specular` were removed, since the shader now fills the corresponding built-in AOVs. Parameters `offscreen_color` and `background_type` were removed. Specular reflection is now affected by Fresnel. To roll back to the previous specular behaviour, set `specular_IOR` to a high value like 100, which effectively disables the Fresnel effect. See notes in the enhancements section above.

## Bug fixes

### 5.0.2.0

- #3319 Alpha not fully opaque in output images
- #4559 Non-linkable light colors should have the linkable metadata disabled
- #5974 Distant light not motion blurring direction
- #5981 Incorrect melanin absorption values for wide gamut rendering color spaces
- #6086 procedural memory overhead
- #6087 maketx fails when run on Windows with read only (mandatory) user profiles
- #6089 Deep EXR output of light path expressions missing volumes
- #6094 artifacts in latlong skydome\_light
- #6095 Fresnel discontinuity in diffuse term when texture is connected to `specular_color`
- #6097 Subdiv: duplicate vertex index in face causes crash
- #6104 Quad lights do not work with projected textures
- #6116 Warn that images will be watermarked if license authorization fails
- #6124 mesh\_light crashing when provided non-mesh node
- #6128 curvature shading differences when seen through glossy transmission/reflection
- #6135 crash during accel construction if there are over 65k overlapping primitives
- #6136 Indirect sample clamp not visible in AOVs
- #6138 light groups not supported by volume shapes and surface shader interiors
- #6141 "A" AOV is black when 8 light AOVs are used
- #6143 AiShaderGlobalsGetVertexUVs not working for uvsets in free render mode
- #6151 Black AOV output due to conflicting AOV type redefinition
- #6162 AiShaderGlobalsGetPositionAtTime, AiTraceBackground, and AiVolumeSample crash in background shading context
- #6166 AiM4Scaling does not work in python
- #6168 Overriding view direction for metal BSDF not supported
- #6170 Crash creating motion blurred `min_pixel_width` inside a procedural
- #6177 wireframe artifacts when not in raster-space
- #6182 MotionVector AOV empty for negative motion start / end
- #6184 MtoA crash when saving a scene using motion blur
- #6186 triplanar uses wrong mipmap level for some parts of the object
- #6187 triplanar uses overly high res mipmaps

- #6192 Crash when removing and recreating a procedural instance
- #6202 counter overflow crash in big scanline EXR images
- #6203 Camera corruption after substantial forward zooming with kick -ipr m
- #6219 photometric\_light filename does not support environment variable expansion
- #6222 Metallic BSDF albedo is always white
- #6242 Crash when removing a procedural node
- #6245 shadow\_matte AOVs
- #6248 Remove offscreen\_color from shadow\_matte
- #6249 Remove shadow\_matte background\_type
- #6250 Re-introduce background\_color in shadow\_matte
- #6251 shadow\_matte self-reflections / self-shadowing
- #6254 trace\_set: crash when destroying shader
- #6255 sss\_irradiance\_shader fails with closure-based shaders
- #6256 Add alpha\_mask in shadow\_matte
- #6286 Volume shader: crash when connected as atmosphere shader
- #6287 Crash with invalid options.atmosphere and options.background shaders
- #6288 Miscellaneous crashes (empty nurbs, empty implicit, atmosphere shadow\_matte)
- #6292 Deep Driver: write errors hang the render
- #6293 Deep driver: append does not work with overscan renders
- #6294 Bucket call back: user bucket coords should be snapped to bucket grid
- #6295 Crash when saving empty ginstance with open\_procs enabled
- #6297 Camera differential evaluation can cause crashes or hangs with OSL shaders linked to camera
- #6313 crash when 4-channel opacity texture has no alpha=0
- #6331 Crash on Windows when loading plugins without .dll extension
- #6353 barndoors light filter result not order-independent
- #6174 "host app" metadata item not readable by AiMetadataStoreLoadFromASS
- #6258 Range shader can sometimes output infinite or nan
- #6303 Support upper and mixed case versions of .ass / .ass.gz extensions

## 5.0.2.3

### 5.0.2.3

- #6511 UDIMs: incorrect character substitutions
- #6481 crash during accel construction with invalid or invisible geometry
- #6500 error building BVH over procedurals with no visible objects
- #6516 Properly warn when there are more than 15 per-light AOVs
- #6527maketx and AiMakeTx are extremely slow or crash with certain large images

## 5.0.2.2

- #6257 s/twrap not working with <attr> expressions set in the image node filename
- #6411 Excessive memory use with compressed UVs
- #6417 Small PolyMesh memory leak at render shutdown
- #6422 OSL raytype mismatch
- #6440 Handle invalid characters when image tokens are resolved
- #6441 Backfacing normals not supported in normal\_map
- #6442 Chained bumps using bump2d isn't working
- #6453 Custom procedurals cannot create nodes before proc\_init
- #6454 Crash With Linear Subdivision, Vertex Normals, and Deformation Blur

## 5.0.2.1

- #6409 ASS metadata load is too slow
- #6412 Crash when setting an array user parameter with its current value

## 5.0.2.0

### Enhancements

- **New sub-surface scattering algorithm:** A new, more accurate way of calculating SSS has been added. Unlike the current empirical BSSRDF method based on diffusion theory, this new method actually traces below the surface with a real random walk and makes no assumptions about the geometry being locally flat. This means it can take into account anisotropic scattering like brute-force volume rendering and produces much better results around concavities and small details. It can also be substantially faster for large scattering radius (i.e. large mean free path) compared to the old method. On the other hand, the new method can be slower in dense media (i.e. small mfp), does not support `sss_setname` for blending two surfaces together, may require redialing materials to achieve a similar look, and is more sensitive to non-closed meshes, "mout bags", and internal geometry potentially casting shadows. This new algorithm is exposed in the `standard_surface` shader via the new parameters `subsurface_type` (with enum values `diffusion` and `randomwalk`) and `subsurface_anisotropy` (Henyey-Greenstein's eccentricity `g` from -1.0 to +1.0). The default is to use the old empirical diffusion method in order not to break the look of existing scenes.
- **Car paint shader:** We are now shipping a dedicated shader for car paint, which can be thought of as the combination of a simplified version of the `standard_surface` and `flakes` shaders. This shader can create a wide range of car paint looks without having to connect several nodes. For example, a pearlescent effect can be easily added to both the specular and flakes layers by simply tweaking a few parameters such as `specular_flip_flop` and `flake_flip_flop`. An arbitrary number of layers of flakes can be used (`flake_layers`). The flakes at a deep layer are covered by the ones closer to the surface and more tinted by pigments (specified by the `transmission_color` parameter).
- **Subdivision frustum culling:** Subdivision patches outside the view or dicing camera frustum will not be subdivided. This is useful for any extended surface that is only partially visible as only the directly visible part will be subdivided. Similarly, no subdivision work will happen if a mesh is not directly visible. This can be turned on globally by setting `options.subdiv_frustum_culling true`

and can be turned off for specific meshes with `polymesh.subdiv_frustum_ignore true`. The global `options.subdiv_frustum_padding` adds a world space padding to the frustum that can be increased as needed to minimize artifacts from out-of-view objects in cast shadows, reflections, etc. Note that motion blur is not yet taken into account and moving objects might require some additional padding.

- **Improved accuracy of UV coords:** Mesh UV coordinates are now handled with higher numerical precision, fixing jagged artifacts that sometimes appeared when using high-resolution UDIM textures over very wide UV ranges.
- **Improved volume sampling of low-spread lights:** Low-spread quad and disk lights should now produce less noise and show increased performance when participating in atmospheric scattering effects.
- **Improved procedural namespace memory usage:** Reduced memory used by procedural namespacing by about 28KB per procedural primitive, so that procedurals now use very little additional memory. In a scene with 100K procedurals, this gave about a 2.7GB reduction in memory use.
- **Faster opacity masks:** Texture map-based opacity masks will now be faster to render. Testing indicates around 3-13% faster.
- **Faster IPR:** The message logging system has been optimized, resulting in about 10% faster performance in IPR, like while moving the camera. In addition, the IPR mode in `kick` has been made substantially more responsive.
- **Faster `AiNodeDestroy`:** we have optimized removal time for nodes contained in procedurals, greatly reducing the shutdown time at the end of a render in complex scenes.
- **Faster `triplanar` shader:** Texture filtering in the `triplanar` shader has been improved, giving better antialiasing and up to a 2x speedup, specially when using high-resolution texture maps.
- **Cellular option in `triplanar` shader:** The `triplanar` shader now supports projection through Voronoi cells using the new `cell` parameter. The rotation angle of the projected texture for each cell can be controlled with the `cell_rotate` parameter. Cells can be smoothly blended using the `cell_blend` parameter.
- **Improved `flakes` shader:** The `size` parameter is replaced by the `density` parameter, which makes it easy to control the size and number of flakes. Alpha channel can be used as a mask. The new `flakes` shader supports non-disc shapes and 3D flakes, which are useful to render gemstone inclusions like goldstone, for example.
- **Improved `shadow_matte` shader:** We have revamped and simplified the shader to make it easier to use, and fixed a number of long-standing issues: Indirect illumination now fills the global `diffuse_indirect` and `specular_indirect` AOVs, so we have removed the shader's (confusingly named) `indirect_diffuse` and `indirect_specular` AOVs. Self-reflections are no longer rendered. A new `specular_IOR` parameter was added that controls Fresnel reflection. Parameters `offscreen_color` and `background_round_type` were removed. The new enum parameter `background` can be set to either `scene_background` (default) or `background_color`, which allows to connect a specific texture in the `background_color` parameter slot. Parameter `alpha_mask` was added to control whether the alpha must be opaque or if it has to contain the shadow mask.
- **Support for more OSL attributes:** OSL shaders now support `getattribute()` lookups of standard camera attributes (e.g. `camera:fov`, `camera:resolution`, etc) as well as the geometry attributes `geom:type`, `geom:name`, `geom:bounds`, and `geom:objbounds` on objects.
- **Transmit AOVs and Alpha:** The `standard_surface` shader with transmission can now pass through AOVs, by enabling the `transmit_aovs` parameter. If the background is transparent, then the transmissive surface will become transparent so that it can be composited over another background. Light path expression AOVs will be passed through, so that for example a diffuse surface seen through a transmissive surface will end up in the `diffuse` AOV. Other AOVs can also be passed straight through (without any opacity blending), which can be used for creating masks for example.
- **Improved multi-threaded render time stats:** Render times when using more than one thread did not really work. We have improved this so that render times are now much more reliable and useful and can now be confidently used to determine what parts of Arnold are the most expensive. In particular, the subdivision and displacement times will now show how much total time was used as well as what fraction of that time was spent with threads unable to do useful work. This "threads blocked" time can often be lowered by using larger buckets or the `random bucket_scanning` ordering.
- **Custom procedural namespaces:** Procedurals can now declare a custom namespace using the new `namespace` parameter. This custom namespace can be used instead of the procedural name, to reference contents through absolute or relative paths. Multiple procedurals can share the same namespace by using the same custom name. Also, they can declare an empty name and they will use the global namespace. (#6085)
- **Added `-turn_smooth` option to `kick`:** When using `kick -turn`, the `-turn_smooth` flag can be added to smoothly start and stop the movement as the original position is reached with a cubic ramp.
- **Added `-laoVs` option to `kick`:** Using `kick -laoVs file.ass` will display a list of all the built-in AOVs and all the AOVs registered by this .ass file.
- **Added `cputime heatmap` view to `kick`:** When using `kick` you can now toggle between viewing kicks default output and a cputime heatmap with the T key. The mapping of the heat map can be scaled with the [ and ] keys.
- **Support for wasd keys in `kick -ipr m`:** Running `kick` in Maya-style IPR mode with `-ipr m` now supports "wasd" style keyboard movement. This was previously only available in Quake-style mode, `-ipr q`.
- **`maketx` version info:** The custom `maketx` binary that ships with Arnold now reports, both in the command-line and in the embedded EXR headers, that it was built specifically for "OpenImageIO-Arnold", to distinguish it from the official "OpenImageIO" one.
- **`maketx` color spaces:** The custom `maketx` that ships with Arnold now supports OCIO and SynColor (when available) through the `colorengine`, `colorconfig` and `colorconvert` flags. See `maketx --help`.
- **`AiMakeTx` reports info messages:** `AiMakeTx` now also prints out OIIO informational messages (generated in verbose mode, for instance) in addition to errors.
- **`AiMakeTx` releases input file lock sooner:** `AiMakeTx` will now close the input texture as soon as possible instead of waiting for all the `maketx` jobs to finish.
- **`AiMakeTx` and `maketx` optimized flags:** Both `AiMakeTx` and `maketx` now always run with the flags `--monochrome-detect` `--opaque-detect` `--constant-color-detect` `--fixnan` `box3` `--oio`, which can result in smaller .tx files that are faster

to load and take less memory.

- **Report when textures are changed during render:** The log files now report when texture modifications during a render cause a texture read error, which can happen in certain pipelines.
- **OCIO color space family support:** The OCIO color manager now implements color space enumeration by family. This is useful for UI drop down organization.
- **OCIO view/display enumeration:** The OCIO color manager can now enumerate view/display combinations using the "View (Display)" family. This lets client programs filter color spaces when only a display transform is appropriate.
- **.ass metadata from compressed files:** We now support loading metadata from .ass.gz files through the AiMetadataStoreLoadFromASS() API function.
- **Upgraded to OIIO 1.7.17:** OpenImageIO has been upgraded to OIIO 1.7.17.

## API additions

- **Color space family enumeration:** Existing color space families for the current config can be enumerated using the new API methods AiColorManagerGetNumFamilies and AiColorManagerGetFamilyNameByIndex. The addition of these new API methods requires any existing custom color managers (which we know are very rare) to be recompiled.
- **AiAOVSsampleIteratorGetPixel()**: Custom filters can now determine what pixel is being filtered with the new API method AiAOVSsampleIteratorGetPixel().
- **transparent LPE label**: When setting the transparent LPE label on a BSDF, the surface will act as if it is transparent and pass through AOVs. This would typically be used for transmission BSDFs, as it is in the standard\_surface shader.
- **Deprecated API warnings**: Defining AI\_ENABLE\_DEPRECATED\_WARNINGS will cause the compiler to emit warnings if deprecated Arnold API functions are used.
- **Random walk SSS closure**: The new random walk SSS algorithm is exposed in the C++ API as AiClosureRandomWalkBSSRDF(). The corresponding OSL closure is randomwalk\_bssrdf.

## Incompatible changes

- **autotile disabled by default**: We found that the autotile code in OIIO does not scale with high-resolution textures. In order to avoid very slow loading of untextured (such as JPEG) when autotile was enabled, we have now changed the autotile default setting to 0, which effectively disables it. In very rare cases, when rendering with a large number of high-resolution untextured textures, this change might degrade performance as the texture cache will blow up. The real solution is to never use untextured files, and instead convert all untextured textures to .tiled files.
- **maketx dependencies**: The custom maketx that ships with Arnold now depends dynamically on `libai.so` and optionally on `sync_color_shader.so`, therefore to work correctly it needs to be run from its installation folder, like `kick`, or alternatively the new dependencies should be copied to where `maketx` is running from.
- **Light groups and volume shading**: Just like with surface shapes, Arnold now obeys light group assignment on volume shapes and surfaces with volumetric interiors. While in many cases desirable, this can produce an unexpected change in the final image in scenes with light group assignments.
- **flakes shader**: It was not easy to control the number of flakes with the `size` and `scale` parameters because they were mutually dependent. Now this can be easily done using the new `density` parameter. The shape of each flake has been changed from disc to Voronoi cell, which is more suitable to render inclusions of gem stones. The shader output type has been changed from RGB to RGBA to support a mask.
- **motionvector AOV**: The motion vector scaling factor in the built-in `motionvector` AOV has changed. This was required to fix a bug that caused zero motion vectors for certain shutter positions. The output from the `motion_vector` shader is unchanged: it can be used as a workaround in your old scenes if you require the previous scaling.
- **shadow\_matte changes**: The shader AOVs `indirect_diffuse` and `indirect_specular` were removed, since the shader now fills the corresponding built-in AOVs. Parameters `offscreen_color` and `background_type` were removed. Specular reflection is now affected by Fresnel. To roll back to the previous specular behaviour, set `specular_IOR` to a high value like 100, which effectively disables the Fresnel effect. See notes in the enhancements section above.

## Bug fixes

### 5.0.2.0

- #3319 Alpha not fully opaque in output images
- #4559 Non-linkable light colors should have the linkable metadata disabled
- #5974 Distant light not motion blurring direction
- #5981 Incorrect melanin absorption values for wide gamut rendering color spaces
- #6086 procedural memory overhead

- #6087 maketx fails when run on Windows with read only (mandatory) user profiles
- #6089 Deep EXR output of light path expressions missing volumes
- #6094 artifacts in latlong skydome\_light
- #6095 Fresnel discontinuity in diffuse term when texture is connected to specular\_color
- #6097 Subdiv: duplicate vertex index in face causes crash
- #6104 Quad lights do not work with projected textures
- #6116 Warn that images will be watermarked if license authorization fails
- #6124 mesh\_light crashing when provided non-mesh node
- #6128 curvature shading differences when seen through glossy transmission/reflection
- #6135 crash during accel construction if there are over 65k overlapping primitives
- #6136 Indirect sample clamp not visible in AOVs
- #6138 light groups not supported by volume shapes and surface shader interiors
- #6141 "A" AOV is black when 8 light AOVs are used
- #6143 AiShaderGlobalsGetVertexUVs not working for uvsets in free render mode
- #6151 Black AOV output due to conflicting AOV type redefinition
- #6162 AiShaderGlobalsGetPositionAtTime, AiTraceBackground, and AiVolumeSample crash in background shading context
- #6166 AiM4Scaling does not work in python
- #6168 Overriding view direction for metal BSDF not supported
- #6170 Crash creating motion blurred min\_pixel\_width inside a procedural
- #6177 wireframe artifacts when not in raster-space
- #6182 MotionVector AOV empty for negative motion start / end
- #6184 MtoA crash when saving a scene using motion blur
- #6186 triplanar uses wrong mipmap level for some parts of the object
- #6187 triplanar uses overly high res mipmaps
- #6192 Crash when removing and recreating a procedural instance
- #6202 counter overflow crash in big scanline EXR images
- #6203 Camera corruption after substantial forward zooming with kick -ipr m
- #6219 photometric\_light filename does not support environment variable expansion
- #6222 Metallic BSDF albedo is always white
- #6242 Crash when removing a procedural node
- #6245 shadow\_matte AOVs
- #6248 Remove offscreen\_color from shadow\_matte
- #6249 Remove shadow\_matte background\_type
- #6250 Re-introduce background\_color in shadow\_matte
- #6251 shadow\_matte self-reflections / self-shadowing
- #6254 trace\_set: crash when destroying shader
- #6255 sss\_irradiance\_shader fails with closure-based shaders
- #6256 Add alpha\_mask in shadow\_matte
- #6286 Volume shader: crash when connected as atmosphere shader

- #6287 Crash with invalid options.atmosphere and options.background shaders
- #6288 Miscellaneous crashes (empty nurbs, empty implicit, atmosphere shadow\_matte)
- #6292 Deep Driver: write errors hang the render
- #6293 Deep driver: append does not work with overscan renders
- #6294 Bucket call back: user bucket coords should be snapped to bucket grid
- #6295 Crash when saving empty ginstance with open\_procs enabled
- #6297 Camera differential evaluation can cause crashes or hangs with OSL shaders linked to camera
- #6313 crash when 4-channel opacity texture has no alpha=0
- #6331 Crash on Windows when loading plugins without .dll extension
- #6353 barndoors light filter result not order-independent
- #6174 "host app" metadata item not readable by AiMetadataStoreLoadFromASS
- #6258 Range shader can sometimes output infinite or nan
- #6303 Support upper and mixed case versions of .ass / .ass.gz extensions

## 5.0.2.2

- #6257 s/twrap not working with <attr> expressions set in the image node filename
- #6411 Excessive memory use with compressed UVs
- #6417 Small PolyMesh memory leak at render shutdown
- #6422 OSL raytype mismatch
- #6440 Handle invalid characters when image tokens are resolved
- #6441 Backfacing normals not supported in normal\_map
- #6442 Chained bumps using bump2d isn't working
- #6453 Custom procedurals cannot create nodes before proc\_init
- #6454 Crash With Linear Subdivision, Vertex Normals, and Deformation Blur

## 5.0.2.1

- #6409 ASS metadata load is too slow
- #6412 Crash when setting an array user parameter with its current value

## 5.0.2.0

### Enhancements

- **New sub-surface scattering algorithm:** A new, more accurate way of calculating SSS has been added. Unlike the current empirical BSSRDF method based on diffusion theory, this new method actually traces below the surface with a real random walk and makes no assumptions about the geometry being locally flat. This means it can take into account anisotropic scattering like brute-force volume rendering and produces much better results around concavities and small details. It can also be substantially faster for large scattering radius (i.e. large mean free path) compared to the old method. On the other hand, the new method can be slower in dense media (i.e. small mfp), does not support `sss_setname` for blending two surfaces together, may require redialing materials to achieve a similar look, and is more sensitive to non-closed meshes, "mouth bags", and internal geometry potentially casting shadows. This new algorithm is exposed in the `standard_surface` shader via the new parameters `subsurface_type` (with enum values `diffusion` and `randomwalk`) and `subsurface_anisotropy` (Henyey-Greenstein's eccentricity `g` from -1.0 to +1.0). The default is to use the old empirical diffusion method in order not to break the look of existing scenes.
- **Car paint shader:** We are now shipping a dedicated shader for car paint, which can be thought of as the combination of a simplified version of the `standard_surface` and `flakes` shaders. This shader can create a wide range of car paint looks without having to connect several nodes. For example, a pearlescent effect can be easily added to both the specular and flakes layers by simply tweaking a few parameters such as `specular_flip_flop` and `flake_flip_flop`. An arbitrary number of layers of flakes can be used (`flake_layers`). The flakes at a deep layer are covered by the ones closer to the surface and more tinted by pigments (specified by the `transmission_color` parameter).
- **Subdivision frustum culling:** Subdivision patches outside the view or dicing camera frustum will not be subdivided. This is useful for any extended surface that is only partially visible as only the directly visible part will be subdivided. Similarly, no subdivision work will happen if a mesh is not directly visible. This can be turned on globally by setting `options.subdiv_frustum_culling true` and can be turned off for specific meshes with `polymesh.subdiv_frustum_ignore true`. The global `options.subdiv_frustum_padding` adds a world space padding to the frustum that can be increased as needed to minimize artifacts from out-of-view objects in cast shadows, reflections, etc. Note that motion blur is not yet taken into account and moving objects might require some additional padding.
- **Improved accuracy of UV coords:** Mesh UV coordinates are now handled with higher numerical precision, fixing jagged artifacts that sometimes appeared when using high-resolution UDIM textures over very wide UV ranges.
- **Improved volume sampling of low-spread lights:** Low-spread quad and disk lights should now produce less noise and show increased performance when participating in atmospheric scattering effects.
- **Improved procedural namespace memory usage:** Reduced memory used by procedural namespacing by about 28KB per procedural primitive, so that procedurals now use very little additional memory. In a scene with 100K procedurals, this gave about a 2.7GB reduction in memory use.
- **Faster opacity masks:** Texture map-based opacity masks will now be faster to render. Testing indicates around 3-13% faster.
- **Faster IPR:** The message logging system has been optimized, resulting in about 10% faster performance in IPR, like while moving the camera. In addition, the IPR mode in `kick` has been made substantially more responsive.

- **Faster `AiNodeDestroy`:** we have optimized removal time for nodes contained in procedurals, greatly reducing the shutdown time at the end of a render in complex scenes.
- **Faster `triplanar shader`:** Texture filtering in the `triplanar` shader has been improved, giving better antialiasing and up to a 2x speedup, specially when using high-resolution texture maps.
- **Cellular option in `triplanar shader`:** The `triplanar` shader now supports projection through Voronoi cells using the new `cell` parameter. The rotation angle of the projected texture for each cell can be controlled with the `cell_rotate` parameter. Cells can be smoothly blended using the `cell_blend` parameter.
- **Improved `flakes shader`:** The `size` parameter is replaced by the `density` parameter, which makes it easy to control the size and number of flakes. Alpha channel can be used as a mask. The new `flakes` shader supports non-disc shapes and 3D flakes, which are useful to render gemstone inclusions like goldstone, for example.
- **Improved `shadow_matte shader`:** We have revamped and simplified the shader to make it easier to use, and fixed a number of long-standing issues: Indirect illumination now fills the global `diffuse_indirect` and `specular_indirect` AOVs, so we have removed the shader's (confusingly named) `indirect_diffuse` and `indirect_specular` AOVs. Self-reflections are no longer rendered. A new `specular_IOR` parameter was added that controls Fresnel reflection. Parameters `offscreen_color` and `background_type` were removed. The new enum parameter `background` can be set to either `scene_background` (default) or `background_color`, which allows to connect a specific texture in the `background_color` parameter slot. Parameter `alpha_mask` was added to control whether the alpha must be opaque or if it has to contain the shadow mask.
- **Support for more OSL attributes:** OSL shaders now support `getattribute()` lookups of standard camera attributes (e.g. `camera:fov`, `camera:resolution`, etc) as well as the geometry attributes `geom:type`, `geom:name`, `geom:bounds`, and `geom:objbounds` on objects.
- **Transmit AOVs and Alpha:** The `standard_surface` shader with transmission can now pass through AOVs, by enabling the `transmit_aovs` parameter. If the background is transparent, then the transmissive surface will become transparent so that it can be composited over another background. Light path expression AOVs will be passed through, so that for example a diffuse surface seen through a transmissive surface will end up in the `diffuse` AOV. Other AOVs can also be passed straight through (without any opacity blending), which can be used for creating masks for example.
- **Improved multi-threaded render time stats:** Render times when using more than one thread did not really work. We have improved this so that render times are now much more reliable and useful and can now be confidently used to determine what parts of Arnold are the most expensive. In particular, the subdivision and displacement times will now show how much total time was used as well as what fraction of that time was spent with threads unable to do useful work. This "threads blocked" time can often be lowered by using larger buckets or the `random bucket_scanning` ordering.
- **Custom procedural namespaces:** Procedurals can now declare a custom namespace using the new `namespace` parameter. This custom namespace can be used instead of the procedural name, to reference contents through absolute or relative paths. Multiple procedurals can share the same namespace by using the same custom name. Also, they can declare an empty name and they will use the global namespace. (#6085)
- **Added `-turn_smooth` option to `kick`:** When using `kick -turn`, the `-turn_smooth` flag can be added to smoothly start and stop the movement as the original position is reached with a cubic ramp.
- **Added `-laoVs` option to `kick`:** Using `kick -laoVs file.ass` will display a list of all the built-in AOVs and all the AOVs registered by this .ass file.
- **Added `cputime heatmap` view to `kick`:** When using `kick` you can now toggle between viewing kicks default output and a cputime heatmap with the T key. The mapping of the heat map can be scaled with the [ and ] keys.
- **Support for wasd keys in `kick -ipr m`:** Running `kick` in Maya-style IPR mode with `-ipr m` now supports "wasd" style keyboard movement. This was previously only available in Quake-style mode, `-ipr q`.
- **`maketx` version info:** The custom `maketx` binary that ships with Arnold now reports, both in the command-line and in the embedded EXR headers, that it was built specifically for "OpenImageIO-Arnold", to distinguish it from the official "OpenImageIO" one.
- **`maketx` color spaces:** The custom `maketx` that ships with Arnold now supports OCIO and SynColor (when available) through the `colorengine`, `colorconfig` and `colorconvert` flags. See `maketx --help`.
- **`AiMakeTx` reports info messages:** `AiMakeTx` now also prints out OIIO informational messages (generated in verbose mode, for instance) in addition to errors.
- **`AiMakeTx` releases input file lock sooner:** `AiMakeTx` will now close the input texture as soon as possible instead of waiting for all the `maketx` jobs to finish.
- **`AiMakeTx` and `maketx` optimized flags:** Both `AiMakeTx` and `maketx` now always run with the flags `--monochrome-detect` `--opaque-detect` `--constant-color-detect` `--fixnan` `box3` `--oio`, which can result in smaller .tx files that are faster to load and take less memory.
- **Report when textures are changed during render:** The log files now report when texture modifications during a render cause a texture read error, which can happen in certain pipelines.
- **OCIO color space family support:** The OCIO color manager now implements color space enumeration by family. This is useful for UI drop down organization.
- **OCIO view/display enumeration:** The OCIO color manager can now enumerate view/display combinations using the "View (Display)" family. This lets client programs filter color spaces when only a display transform is appropriate.
- **.ass metadata from compressed files:** We now support loading metadata from .ass.gz files through the `AiMetadataStoreLoadFromASS()` API function.
- **Upgraded to OIIO 1.7.17:** OpenImageIO has been upgraded to OIIO 1.7.17.

## API additions

- **Color space family enumeration:** Existing color space families for the current config can be enumerated using the new API methods `AiColorManagerGetNumFamilies` and `AiColorManagerGetFamilyNameByIndex`. The addition of these new API methods requires any existing custom color managers (which we know are very rare) to be recompiled.
- **`AiAOVSampleIteratorGetPixel()`:** Custom filters can now determine what pixel is being filtered with the new API method `AiAOVSampleIteratorGetPixel()`.
- **transparent LPE label:** When setting the `transparent` LPE label on a BSDF, the surface will act as if it is transparent and pass through AOVs. This would typically be used for transmission BSDFs, as it is in the `standard_surface` shader.
- **Deprecated API warnings:** Defining `AI_ENABLE_DEPRECATED_WARNINGS` will cause the compiler to emit warnings if deprecated Arnold API functions are used.
- **Random walk SSS closure:** The new random walk SSS algorithm is exposed in the C++ API as `AiClosureRandomWalkBSSRDF()`. The corresponding OSL closure is `randomwalk_bssrdf`.

## Incompatible changes

- **autotile disabled by default:** We found that the `autotile` code in OIIO does not scale with high-resolution textures. In order to avoid very slow loading of untextured textures (such as JPEG) when `autotile` was enabled, we have now changed the `autotile` default setting to 0, which effectively disables it. In very rare cases, when rendering with a large number of high-resolution untextured textures, this change might degrade performance as the texture cache will blow up. The real solution is to never use untextured files, and instead convert all untextured textures to .tx tiled files.
- **`maketx` dependencies:** The custom `maketx` that ships with Arnold now depends dynamically on `libai.so` and optionally on `sync_color_shader.so`, therefore to work correctly it needs to be run from its installation folder, like `kick`, or alternatively the new dependencies should be copied to where `maketx` is running from.
- **Light groups and volume shading:** Just like with surface shapes, Arnold now obeys light group assignment on volume shapes and surfaces with volumetric interiors. While in many cases desirable, this can produce an unexpected change in the final image in scenes with light group assignments.
- **`flakes` shader:** It was not easy to control the number of flakes with the `size` and `scale` parameters because they were mutually dependent. Now this can be easily done using the new `density` parameter. The shape of each flake has been changed from disc to Voronoi cell, which is more suitable to render inclusions of gem stones. The shader output type has been changed from RGB to RGBA to support a mask.
- **`motionvector` AOV:** The motion vector scaling factor in the built-in `motionvector` AOV has changed. This was required to fix a bug that caused zero motion vectors for certain shutter positions. The output from the `motion_vector` shader is unchanged: it can be used as a workaround in your old scenes if you require the previous scaling.
- **`shadow_mattes` changes:** The shader AOVs `indirect_diffuse` and `indirect_specular` were removed, since the shader now fills the corresponding built-in AOVs. Parameters `offscreen_color` and `background_type` were removed. Specular reflection is now affected by Fresnel. To roll back to the previous specular behaviour, set `specular_IOR` to a high value like 100, which effectively disables the Fresnel effect. See notes in the enhancements section above.

## Bug fixes

### 5.0.2.0

- #3319 Alpha not fully opaque in output images
- #4559 Non-linkable light colors should have the linkable metadata disabled
- #5974 Distant light not motion blurring direction
- #5981 Incorrect melanin absorption values for wide gamut rendering color spaces
- #6086 procedural memory overhead
- #6087 `maketx` fails when run on Windows with read only (mandatory) user profiles
- #6089 Deep EXR output of light path expressions missing volumes
- #6094 artifacts in latlong `skydome_light`
- #6095 Fresnel discontinuity in diffuse term when texture is connected to `specular_color`
- #6097 Subdiv: duplicate vertex index in face causes crash
- #6104 Quad lights do not work with projected textures
- #6116 Warn that images will be watermarked if license authorization fails
- #6124 `mesh_light` crashing when provided non-mesh node

- #6128 curvature shading differences when seen through glossy transmission/reflection
- #6135 crash during accel construction if there are over 65k overlapping primitives
- #6136 Indirect sample clamp not visible in AOVs
- #6138 light groups not supported by volume shapes and surface shader interiors
- #6141 "A" AOV is black when 8 light AOVs are used
- #6143 AiShaderGlobalsGetVertexUVs not working for uvsets in free render mode
- #6151 Black AOV output due to conflicting AOV type redefinition
- #6162 AiShaderGlobalsGetPositionAtTime, AiTraceBackground, and AiVolumeSample crash in background shading context
- #6166 AiM4Scaling does not work in python
- #6168 Overriding view direction for metal BSDF not supported
- #6170 Crash creating motion blurred min\_pixel\_width inside a procedural
- #6177 wireframe artifacts when not in raster-space
- #6182 MotionVector AOV empty for negative motion start / end
- #6184 MtoA crash when saving a scene using motion blur
- #6186 triplanar uses wrong mipmap level for some parts of the object
- #6187 triplanar uses overly high res mipmaps
- #6192 Crash when removing and recreating a procedural instance
- #6202 counter overflow crash in big scanline EXR images
- #6203 Camera corruption after substantial forward zooming with kick -ipr m
- #6219 photometric\_light filename does not support environment variable expansion
- #6222 Metallic BSDF albedo is always white
- #6242 Crash when removing a procedural node
- #6245 shadow\_matte AOVs
- #6248 Remove offscreen\_color from shadow\_matte
- #6249 Remove shadow\_matte background\_type
- #6250 Re-introduce background\_color in shadow\_matte
- #6251 shadow\_matte self-reflections / self-shadowing
- #6254 trace\_set: crash when destroying shader
- #6255 sss\_irradiance\_shader fails with closure-based shaders
- #6256 Add alpha\_mask in shadow\_matte
- #6286 Volume shader: crash when connected as atmosphere shader
- #6287 Crash with invalid options.atmosphere and options.background shaders
- #6288 Miscellaneous crashes (empty nurbs, empty implicit, atmosphere shadow\_matte)
- #6292 Deep Driver: write errors hang the render
- #6293 Deep driver: append does not work with overscan renders
- #6294 Bucket call back: user bucket coords should be snapped to bucket grid
- #6295 Crash when saving empty ginstance with open\_procs enabled
- #6297 Camera differential evaluation can cause crashes or hangs with OSL shaders linked to camera
- #6313 crash when 4-channel opacity texture has no alpha=0

- #6331 Crash on Windows when loading plugins without .dll extension
- #6353 barndoors light filter result not order-independent
- #6174 "host app" metadata item not readable by AiMetadataStoreLoadFromASS
- #6258 Range shader can sometimes output infinite or nan
- #6303 Support upper and mixed case versions of .ass / .ass.gz extensions

## 5.0.2.1

### 5.0.2.1

- #6409 ASS metadata load is too slow
- #6412 Crash when setting an array user parameter with its current value

## 5.0.2.0

### Enhancements

- **New sub-surface scattering algorithm:** A new, more accurate way of calculating SSS has been added. Unlike the current empirical BSSRDF method based on diffusion theory, this new method actually traces below the surface with a real random walk and makes no assumptions about the geometry being locally flat. This means it can take into account anisotropic scattering like brute-force volume rendering and produces much better results around concavities and small details. It can also be substantially faster for large scattering radius (i.e. large mean free path) compared to the old method. On the other hand, the new method can be slower in dense media (i.e. small mfp), does not support `sss_setname` for blending two surfaces together, may require redialing materials to achieve a similar look, and is more sensitive to non-closed meshes, "mouth bags", and internal geometry potentially casting shadows. This new algorithm is exposed in the `standard_surface` shader via the new parameters `subsurface_type` (with enum values `diffusion` and `randomwalk`) and `subsurface_anisotropy` (Henyey-Greenstein's eccentricity `g` from -1.0 to +1.0). The default is to use the old empirical diffusion method in order not to break the look of existing scenes.
- **Car paint shader:** We are now shipping a dedicated shader for car paint, which can be thought of as the combination of a simplified version of the `standard_surface` and `flakes` shaders. This shader can create a wide range of car paint looks without having to connect several nodes. For example, a pearlescent effect can be easily added to both the specular and flakes layers by simply tweaking a few parameters such as `specular_flip_flop` and `flake_flip_flop`. An arbitrary number of layers of flakes can be used (`flake_layers`). The flakes at a deep layer are covered by the ones closer to the surface and more tinted by pigments (specified by the `transmission_color` parameter).
- **Subdivision frustum culling:** Subdivision patches outside the view or dicing camera frustum will not be subdivided. This is useful for any extended surface that is only partially visible as only the directly visible part will be subdivided. Similarly, no subdivision work will happen if a mesh is not directly visible. This can be turned on globally by setting `options.subdiv_frustum_culling true` and can be turned off for specific meshes with `polymesh.subdiv_frustum_ignore true`. The global `options.subdiv_frustum_padding` adds a world space padding to the frustum that can be increased as needed to minimize artifacts from out-of-view objects in cast shadows, reflections, etc. Note that motion blur is not yet taken into account and moving objects might require some additional padding.
- **Improved accuracy of UV coords:** Mesh UV coordinates are now handled with higher numerical precision, fixing jagged artifacts that sometimes appeared when using high-resolution UDIM textures over very wide UV ranges.
- **Improved volume sampling of low-spread lights:** Low-spread quad and disk lights should now produce less noise and show increased performance when participating in atmospheric scattering effects.
- **Improved procedural namespace memory usage:** Reduced memory used by procedural namespacing by about 28KB per procedural primitive, so that procedurals now use very little additional memory. In a scene with 100K procedurals, this gave about a 2.7GB reduction in memory use.
- **Faster opacity masks:** Texture map-based opacity masks will now be faster to render. Testing indicates around 3-13% faster.
- **Faster IPR:** The message logging system has been optimized, resulting in about 10% faster performance in IPR, like while moving the camera. In addition, the IPR mode in `kick` has been made substantially more responsive.
- **Faster AiNodeDestroy:** we have optimized removal time for nodes contained in procedurals, greatly reducing the shutdown time at the end of a render in complex scenes.
- **Faster triplanar shader:** Texture filtering in the `triplanar` shader has been improved, giving better antialiasing and up to a 2x speedup, specially when using high-resolution texture maps.
- **Celular option in triplanar shader:** The `triplanar` shader now supports projection through Voronoi cells using the new `cell` parameter. The rotation angle of the projected texture for each cell can be controlled with the `cell_rotate` parameter. Cells can be smoothly blended using the `cell_blend` parameter.
- **Improved flakes shader:** The `size` parameter is replaced by the `density` parameter, which makes it easy to control the size and number of flakes. Alpha channel can be used as a mask. The new `flakes` shader supports non-disc shapes and 3D flakes, which are useful to render gemstone inclusions like goldstone, for example.
- **Improved shadow\_matte shader:** We have revamped and simplified the shader to make it easier to use, and fixed a number of long-standing issues: Indirect illumination now fills the global `diffuse_indirect` and `specular_indirect` AOVs, so we have removed the shader's (confusingly named) `indirect_diffuse` and `indirect_specular` AOVs. Self-reflections are no longer rendered. A new `specular_IOR` parameter was added that controls Fresnel reflection. Parameters `offscreen_color` and `background_type` were removed. The new enum parameter `background` can be set to either `scene_background` (default) or `background_color`, which allows to connect a specific texture in the `background_color` parameter slot. Parameter `alpha_mask` was added to control whether the alpha must be opaque or if it has to contain the shadow mask.

- **Support for more OSL attributes:** OSL shaders now support `getattribute()` lookups of standard camera attributes (e.g. `camera:fov`, `camera:resolution`, etc) as well as the geometry attributes `geom:type`, `geom:name`, `geom:bounds`, and `geom:objbounds` on objects.
- **Transmit AOVs and Alpha:** The `standard_surface` shader with transmission can now pass through AOVs, by enabling the `transmit_aovs` parameter. If the background is transparent, then the transmissive surface will become transparent so that it can be composited over another background. Light path expression AOVs will be passed through, so that for example a diffuse surface seen through a transmissive surface will end up in the `diffuse` AOV. Other AOVs can also be passed straight through (without any opacity blending), which can be used for creating masks for example.
- **Improved multi-threaded render time stats:** Render times when using more than one thread did not really work. We have improved this so that render times are now much more reliable and useful and can now be confidently used to determine what parts of Arnold are the most expensive. In particular, the subdivision and displacement times will now show how much total time was used as well as what fraction of that time was spent with threads unable to do useful work. This "threads blocked" time can often be lowered by using larger buckets or the `random bucket_scanning` ordering.
- **Custom procedural namespaces:** Procedurals can now declare a custom namespace using the new `namespace` parameter. This custom namespace can be used instead of the procedural name, to reference contents through absolute or relative paths. Multiple procedurals can share the same namespace by using the same custom name. Also, they can declare an empty name and they will use the global namespace. (#6085)
- **Added -turn\_smooth option to kick:** When using `kick -turn`, the `-turn_smooth` flag can be added to smoothly start and stop the movement as the original position is reached with a cubic ramp.
- **Added -laovs option to kick:** Using `kick -laovs file.ass` will display a list of all the built-in AOVs and all the AOVs registered by this .ass file.
- **Added cputime heatmap view to kick:** When using `kick` you can now toggle between viewing kicks default output and a cputime heatmap with the `T` key. The mapping of the heat map can be scaled with the `[` and `]` keys.
- **Support for wasd keys in kick -ipr m:** Running `kick` in Maya-style IPR mode with `-ipr m` now supports "wasd" style keyboard movement. This was previously only available in Quake-style mode, `-ipr q`.
- **maketx version info:** The custom `maketx` binary that ships with Arnold now reports, both in the command-line and in the embedded EXR headers, that it was built specifically for "OpenImageIO-Arnold", to distinguish it from the official "OpenImageIO" one.
- **maketx color spaces:** The custom `maketx` that ships with Arnold now supports OCIO and SynColor (when available) through the `colorengine`, `colorconfig` and `colorconvert` flags. See `maketx --help`.
- **AiMakeTx reports info messages:** `AiMakeTx` now also prints out OIIO informational messages (generated in verbose mode, for instance) in addition to errors.
- **AiMakeTx releases input file lock sooner:** `AiMakeTx` will now close the input texture as soon as possible instead of waiting for all the `maketx` jobs to finish.
- **AiMakeTx and maketx optimized flags:** Both `AiMakeTx` and `maketx` now always run with the flags `--monochrome-detect` `--opaque-detect` `--constant-color-detect` `--fixnan` `box3` `--oio`, which can result in smaller .tx files that are faster to load and take less memory.
- **Report when textures are changed during render:** The log files now report when texture modifications during a render cause a texture read error, which can happen in certain pipelines.
- **OCIO color space family support:** The OCIO color manager now implements color space enumeration by family. This is useful for UI drop down organization.
- **OCIO view/display enumeration:** The OCIO color manager can now enumerate view/display combinations using the "View (Display)" family. This lets client programs filter color spaces when only a display transform is appropriate.
- **.ass metadata from compressed files:** We now support loading metadata from .ass.gz files through the `AiMetadataStoreLoadFromASS()` API function.
- **Upgraded to OIIO 1.7.17:** OpenImageIO has been upgraded to OIIO 1.7.17.

## API additions

- **Color space family enumeration:** Existing color space families for the current config can be enumerated using the new API methods `AiColorManagerGetNumFamilies` and `AiColorManagerGetFamilyNameByIndex`. The addition of these new API methods requires any existing custom color managers (which we know are very rare) to be recompiled.
- **AiAOVSampleIteratorGetPixel():** Custom filters can now determine what pixel is being filtered with the new API method `AiAOVSampleIteratorGetPixel()`.
- **transparent LPE label:** When setting the `transparent` LPE label on a BSDF, the surface will act as if it is transparent and pass through AOVs. This would typically be used for transmission BSDFs, as it is in the `standard_surface` shader.
- **Deprecated API warnings:** Defining `AI_ENABLE_DEPRECATED_WARNINGS` will cause the compiler to emit warnings if deprecated Arnold API functions are used.
- **Random walk SSS closure:** The new random walk SSS algorithm is exposed in the C++ API as `AiClosureRandomWalkBSSRDF()`. The corresponding OSL closure is `randomwalk_bssrdf`.

## Incompatible changes

- **autotile disabled by default:** We found that the autotile code in OIIO does not scale with high-resolution textures. In order to avoid very slow loading of untextured textures (such as JPEG) when autotile was enabled, we have now changed the `autotile` default

setting to 0, which effectively disables it. In very rare cases, when rendering with a large number of high-resolution untextured textures, this change might degrade performance as the texture cache will blow up. The real solution is to never use untextured files, and instead convert all untextured textures to .tx tiled files.

- **maketx dependencies:** The custom maketx that ships with Arnold now depends dynamically on `libai.so` and optionally on `sync_olor_shader.so`, therefore to work correctly it needs to be run from its installation folder, like `kick`, or alternatively the new dependencies should be copied to where `maketx` is running from.
- **Light groups and volume shading:** Just like with surface shapes, Arnold now obeys light group assignment on volume shapes and surfaces with volumetric interiors. While in many cases desirable, this can produce an unexpected change in the final image in scenes with light group assignments.
- **flakes shader:** It was not easy to control the number of flakes with the `size` and `scale` parameters because they were mutually dependent. Now this can be easily done using the new `density` parameter. The shape of each flake has been changed from disc to Voronoi cell, which is more suitable to render inclusions of gem stones. The shader output type has been changed from RGB to RGBA to support a mask.
- **motionvector AOV:** The motion vector scaling factor in the built-in `motionvector` AOV has changed. This was required to fix a bug that caused zero motion vectors for certain shutter positions. The output from the `motion_vector` shader is unchanged: it can be used as a workaround in your old scenes if you require the previous scaling.
- **shadow\_matte changes:** The shader AOVs `indirect_diffuse` and `indirect_specular` were removed, since the shader now fills the corresponding built-in AOVs. Parameters `offscreen_color` and `background_type` were removed. Specular reflection is now affected by Fresnel. To roll back to the previous specular behaviour, set `specular_IOR` to a high value like 100, which effectively disables the Fresnel effect. See notes in the Enhancements section above.

## Bug fixes

### 5.0.2.0

- #3319 Alpha not fully opaque in output images
- #4559 Non-linkable light colors should have the linkable metadata disabled
- #5974 Distant light not motion blurring direction
- #5981 Incorrect melanin absorption values for wide gamut rendering color spaces
- #6086 procedural memory overhead
- #6087 maketx fails when run on Windows with read only (mandatory) user profiles
- #6089 Deep EXR output of light path expressions missing volumes
- #6094 artifacts in latlong skydome\_light
- #6095 Fresnel discontinuity in diffuse term when texture is connected to `specular_color`
- #6097 Subdiv: duplicate vertex index in face causes crash
- #6104 Quad lights do not work with projected textures
- #6116 Warn that images will be watermarked if license authorization fails
- #6124 mesh\_light crashing when provided non-mesh node
- #6128 curvature shading differences when seen through glossy transmission/reflection
- #6135 crash during accel construction if there are over 65k overlapping primitives
- #6136 Indirect sample clamp not visible in AOVs
- #6138 light groups not supported by volume shapes and surface shader interiors
- #6141 "A" AOV is black when 8 light AOVs are used
- #6143 AiShaderGlobalsGetVertexUVs not working for uvsets in free render mode
- #6151 Black AOV output due to conflicting AOV type redefinition
- #6162 AiShaderGlobalsGetPositionAtTime, AiTraceBackground, and AiVolumeSample crash in background shading context
- #6166 AiM4Scaling does not work in python
- #6168 Overriding view direction for metal BSDF not supported

- #6170 Crash creating motion blurred min\_pixel\_width inside a procedural
- #6177 wireframe artifacts when not in raster-space
- #6182 MotionVector AOV empty for negative motion start / end
- #6184 MtoA crash when saving a scene using motion blur
- #6186 triplanar uses wrong mipmap level for some parts of the object
- #6187 triplanar uses overly high res mipmaps
- #6192 Crash when removing and recreating a procedural instance
- #6202 counter overflow crash in big scanline EXR images
- #6203 Camera corruption after substantial forward zooming with kick -ipr m
- #6219 photometric\_light filename does not support environment variable expansion
- #6222 Metallic BSDF albedo is always white
- #6242 Crash when removing a procedural node
- #6245 shadow\_matte AOVs
- #6248 Remove offscreen\_color from shadow\_matte
- #6249 Remove shadow\_matte background\_type
- #6250 Re-introduce background\_color in shadow\_matte
- #6251 shadow\_matte self-reflections / self-shadowing
- #6254 trace\_set: crash when destroying shader
- #6255 sss\_irradiance\_shader fails with closure-based shaders
- #6256 Add alpha\_mask in shadow\_matte
- #6286 Volume shader: crash when connected as atmosphere shader
- #6287 Crash with invalid options.atmosphere and options.background shaders
- #6288 Miscellaneous crashes (empty nurbs, empty implicit, atmosphere shadow\_matte)
- #6292 Deep Driver: write errors hang the render
- #6293 Deep driver: append does not work with overscan renders
- #6294 Bucket call back: user bucket coords should be snapped to bucket grid
- #6295 Crash when saving empty ginstance with open\_procs enabled
- #6297 Camera differential evaluation can cause crashes or hangs with OSL shaders linked to camera
- #6313 crash when 4-channel opacity texture has no alpha=0
- #6331 Crash on Windows when loading plugins without .dll extension
- #6353 barndoors light filter result not order-independent
- #6174 "host app" metadata item not readable by AiMetadataStoreLoadFromASS
- #6258 Range shader can sometimes output infinite or nan
- #6303 Support upper and mixed case versions of .ass / .ass.gz extensions

## 5.0.2.0

### Enhancements

- **New sub-surface scattering algorithm:** A new, more accurate way of calculating SSS has been added. Unlike the current empirical BSSRDF method based on diffusion theory, this new method actually traces below the surface with a real random walk and makes no assumptions about the geometry being locally flat. This means it can take into account anisotropic scattering like brute-force volume rendering and produces much better results around concavities and small details. It can also be substantially faster for large scattering radius (i.e. large mean free path) compared to the old method. On the other hand, the new method can be slower in dense media (i.e. small mfp), does not support `sss_setname` for blending two surfaces together, may require redialing materials to achieve a similar look, and is more sensitive to non-closed meshes, "mouth bags", and internal geometry potentially casting shadows. This new algorithm is exposed in the `standard_surface` shader via the new parameters `subsurface_type` (with enum values `diffusion` and `randomwalk`) and `subsurface_anisotropy` (Henyey-Greenstein's eccentricity `g` from -1.0 to +1.0). The default is to use the old empirical diffusion method in order not to break the look of existing scenes.
- **Car paint shader:** We are now shipping a dedicated shader for car paint, which can be thought of as the combination of a simplified version of the `standard_surface` and `flakes` shaders. This shader can create a wide range of car paint looks without having to connect several nodes. For example, a pearlescent effect can be easily added to both the specular and flakes layers by simply tweaking a few parameters such as `specular_flip_flop` and `flake_flip_flop`. An arbitrary number of layers of flakes can be used (`flake_layers`). The flakes at a deep layer are covered by the ones closer to the surface and more tinted by pigments (specified by the `transmission_color` parameter).
- **Subdivision frustum culling:** Subdivision patches outside the view or dicing camera frustum will not be subdivided. This is useful for any extended surface that is only partially visible as only the directly visible part will be subdivided. Similarly, no subdivision work will happen if a mesh is not directly visible. This can be turned on globally by setting `options.subdiv_frustum_culling true` and can be turned off for specific meshes with `polymesh.subdiv_frustum_ignore true`. The global `options.subdiv_frustum_padding` adds a world space padding to the frustum that can be increased as needed to minimize artifacts from out-of-view objects in cast shadows, reflections, etc. Note that motion blur is not yet taken into account and moving objects might require some additional padding.
- **Improved accuracy of UV coords:** Mesh UV coordinates are now handled with higher numerical precision, fixing jagged artifacts that sometimes appeared when using high-resolution UDIM textures over very wide UV ranges.
- **Improved volume sampling of low-spread lights:** Low-spread quad and disk lights should now produce less noise and show increased performance when participating in atmospheric scattering effects.
- **Improved procedural namespace memory usage:** Reduced memory used by procedural namespacing by about 28KB per procedural primitive, so that procedurals now use very little additional memory. In a scene with 100K procedurals, this gave about a 2.7GB reduction in memory use.
- **Faster opacity masks:** Texture map-based opacity masks will now be faster to render. Testing indicates around 3-13% faster.
- **Faster IPR:** The message logging system has been optimized, resulting in about 10% faster performance in IPR, like while moving the camera. In addition, the IPR mode in `kick` has been made substantially more responsive.
- **Faster AiNodeDestroy:** we have optimized removal time for nodes contained in procedurals, greatly reducing the shutdown time at the end of a render in complex scenes.
- **Faster triplanar shader:** Texture filtering in the `triplanar` shader has been improved, giving better antialiasing and up to a 2x speedup, specially when using high-resolution texture maps.
- **Celular option in triplanar shader:** The `triplanar` shader now supports projection through Voronoi cells using the new `cell` parameter. The rotation angle of the projected texture for each cell can be controlled with the `cell_rotate` parameter. Cells can be smoothly blended using the `cell_blend` parameter.
- **Improved flakes shader:** The `size` parameter is replaced by the `density` parameter, which makes it easy to control the size and number of flakes. Alpha channel can be used as a mask. The new `flakes` shader supports non-disc shapes and 3D flakes, which are useful to render gemstone inclusions like goldstone, for example.
- **Improved shadow\_matte shader:** We have revamped and simplified the shader to make it easier to use, and fixed a number of long-standing issues: Indirect illumination now fills the global `diffuse_indirect` and `specular_indirect` AOVs, so we have removed the shader's (confusingly named) `indirect_diffuse` and `indirect_specular` AOVs. Self-reflections are no longer rendered. A new `specular_IOR` parameter was added that controls Fresnel reflection. Parameters `offscreen_color` and `background_round_type` were removed. The new enum parameter `background` can be set to either `scene_background` (default) or `background_color`, which allows to connect a specific texture in the `background_color` parameter slot. Parameter `alpha_mask` was added to control whether the alpha must be opaque or if it has to contain the shadow mask.
- **Support for more OSL attributes:** OSL shaders now support `getattribute()` lookups of standard camera attributes (e.g. `camera:a:fov`, `camera:resolution`, etc) as well as the geometry attributes `geom:type`, `geom:name`, `geom:bounds`, and `geom:objbounds` on objects.
- **Transmit AOVs and Alpha:** The `standard_surface` shader with transmission can now pass through AOVs, by enabling the `transmit_aovs` parameter. If the background is transparent, then the transmissive surface will become transparent so that it can be composited over another background. Light path expression AOVs will be passed through, so that for example a diffuse surface seen through a transmissive surface will end up in the `diffuse` AOV. Other AOVs can also be passed straight through (without any opacity blending), which can be used for creating masks for example.
- **Improved multi-threaded render time stats:** Render times when using more than one thread did not really work. We have improved this so that render times are now much more reliable and useful and can now be confidently used to determine what parts of Arnold are the most expensive. In particular, the subdivision and displacement times will now show how much total time was used

as well as what fraction of that time was spent with threads unable to do useful work. This "threads blocked" time can often be lowered by using larger buckets or the `random bucket_scanning` ordering.

- **Custom procedural namespaces:** Procedurals can now declare a custom namespace using the new `namespace` parameter. This custom namespace can be used instead of the procedural name, to reference contents through absolute or relative paths. Multiple procedurals can share the same namespace by using the same custom name. Also, they can declare an empty name and they will use the global namespace. (#6085)
- **Added `-turn_smooth` option to `kick`:** When using `kick -turn`, the `-turn_smooth` flag can be added to smoothly start and stop the movement as the original position is reached with a cubic ramp.
- **Added `-laovs` option to `kick`:** Using `kick -laovs file.ass` will display a list of all the built-in AOVs and all the AOVs registered by this `.ass` file.
- **Added `cputime heatmap` view to `kick`:** When using `kick` you can now toggle between viewing kicks default output and a cputime heatmap with the `T` key. The mapping of the heat map can be scaled with the `[` and `]` keys.
- **Support for `wasd` keys in `kick -ipr m`:** Running `kick` in Maya-style IPR mode with `-ipr m` now supports "`wasd`" style keyboard movement. This was previously only available in Quake-style mode, `-ipr q`.
- **`maketx` version info:** The custom `maketx` binary that ships with Arnold now reports, both in the command-line and in the embedded EXR headers, that it was built specifically for "OpenImageIO-Arnold", to distinguish it from the official "OpenImageIO" one.
- **`maketx` color spaces:** The custom `maketx` that ships with Arnold now supports OCIO and SynColor (when available) through the `colorengine`, `colorconfig` and `colorconvert` flags. See `maketx --help`.
- **AiMakeTx reports info messages:** AiMakeTx now also prints out OIIO informational messages (generated in verbose mode, for instance) in addition to errors.
- **AiMakeTx releases input file lock sooner:** AiMakeTx will now close the input texture as soon as possible instead of waiting for all the maketx jobs to finish.
- **AiMakeTx and maketx optimized flags:** Both AiMakeTx and maketx now always run with the flags `--monochrome-detect` `--opaque-detect` `--constant-color-detect` `--fixnan` `box3` `--oio`, which can result in smaller `.tx` files that are faster to load and take less memory.
- **Report when textures are changed during render:** The log files now report when texture modifications during a render cause a texture read error, which can happen in certain pipelines.
- **OCIO color space family support:** The OCIO color manager now implements color space enumeration by family. This is useful for UI drop down organization.
- **OCIO view/display enumeration:** The OCIO color manager can now enumerate view/display combinations using the "View (Display)" family. This lets client programs filter color spaces when only a display transform is appropriate.
- **.ass metadata from compressed files:** We now support loading metadata from `.ass.gz` files through the `AiMetadataStoreLoadFromASS()` API function.
- **Upgraded to OIIO 1.7.17:** OpenImageIO has been upgraded to OIIO 1.7.17.

## API additions

- **Color space family enumeration:** Existing color space families for the current config can be enumerated using the new API methods `AiColorManagerGetNumFamilies` and `AiColorManagerGetFamilyNameByIndex`. The addition of these new API methods requires any existing custom color managers (which we know are very rare) to be recompiled.
- **AiAOVSampleIteratorGetPixel():** Custom filters can now determine what pixel is being filtered with the new API method `AiAOVSampleIteratorGetPixel()`.
- **transparent LPE label:** When setting the `transparent` LPE label on a BSDF, the surface will act as if it is transparent and pass through AOVs. This would typically be used for transmission BSDFs, as it is in the `standard_surface` shader.
- **Deprecated API warnings:** Defining `AI_ENABLE_DEPRECATED_WARNINGS` will cause the compiler to emit warnings if deprecated Arnold API functions are used.
- **Random walk SSS closure:** The new random walk SSS algorithm is exposed in the C++ API as `AiClosureRandomWalkBSSRDF()`. The corresponding OSL closure is `randomwalk_bssrdf`.

## Incompatible changes

- **autotile disabled by default:** We found that the `autotile` code in OIIO does not scale with high-resolution textures. In order to avoid very slow loading of untextured textures (such as JPEG) when `autotile` was enabled, we have now changed the `autotile` default setting to 0, which effectively disables it. In very rare cases, when rendering with a large number of high-resolution untextured textures, this change might degrade performance as the texture cache will blow up. The real solution is to never use untextured files, and instead convert all untextured textures to `.tx` tiled files.
- **maketx dependencies:** The custom `maketx` that ships with Arnold now depends dynamically on `libai.so` and optionally on `sync_olor_shader.so`, therefore to work correctly it needs to be run from its installation folder, like `kick`, or alternatively the new dependencies should be copied to where `maketx` is running from.
- **Light groups and volume shading:** Just like with surface shapes, Arnold now obeys light group assignment on volume shapes and surfaces with volumetric interiors. While in many cases desirable, this can produce an unexpected change in the final image in scenes with light group assignments.
- **flakes shader:** It was not easy to control the number of flakes with the `size` and `scale` parameters because they were mutually dependent. Now this can be easily done using the new `density` parameter. The shape of each flake has been changed from disc to

Voronoi cell, which is more suitable to render inclusions of gem stones. The shader output type has been changed from RGB to RGBA to support a mask.

- **`motionvector` AOV:** The motion vector scaling factor in the built-in `motionvector` AOV has changed. This was required to fix a bug that caused zero motion vectors for certain shutter positions. The output from the `motion_vector` shader is unchanged: it can be used as a workaround in your old scenes if you require the previous scaling.
- **`shadow_matte` changes:** The shader AOVs `indirect_diffuse` and `indirect_specular` were removed, since the shader now fills the corresponding built-in AOVs. Parameters `offscreen_color` and `background_type` were removed. Specular reflection is now affected by Fresnel. To roll back to the previous specular behaviour, set `specular_IOR` to a high value like 100, which effectively disables the Fresnel effect. See notes in the enhancements section above.

## Bug fixes

### 5.0.2.0

- #3319 Alpha not fully opaque in output images
- #4559 Non-linkable light colors should have the linkable metadata disabled
- #5974 Distant light not motion blurring direction
- #5981 Incorrect melanin absorption values for wide gamut rendering color spaces
- #6086 procedural memory overhead
- #6087 maketx fails when run on Windows with read only (mandatory) user profiles
- #6089 Deep EXR output of light path expressions missing volumes
- #6094 artifacts in latlong skydome\_light
- #6095 Fresnel discontinuity in diffuse term when texture is connected to `specular_color`
- #6097 Subdiv: duplicate vertex index in face causes crash
- #6104 Quad lights do not work with projected textures
- #6116 Warn that images will be watermarked if license authorization fails
- #6124 mesh\_light crashing when provided non-mesh node
- #6128 curvature shading differences when seen through glossy transmission/reflection
- #6135 crash during accel construction if there are over 65k overlapping primitives
- #6136 Indirect sample clamp not visible in AOVs
- #6138 light groups not supported by volume shapes and surface shader interiors
- #6141 "A" AOV is black when 8 light AOVs are used
- #6143 AiShaderGlobalsGetVertexUVs not working for uvsets in free render mode
- #6151 Black AOV output due to conflicting AOV type redefinition
- #6162 AiShaderGlobalsGetPositionAtTime, AiTraceBackground, and AiVolumeSample crash in background shading context
- #6166 AiM4Scaling does not work in python
- #6168 Overriding view direction for metal BSDF not supported
- #6170 Crash creating motion blurred `min_pixel_width` inside a procedural
- #6177 wireframe artifacts when not in raster-space
- #6182 MotionVector AOV empty for negative motion start / end
- #6184 MtoA crash when saving a scene using motion blur
- #6186 triplanar uses wrong mipmap level for some parts of the object
- #6187 triplanar uses overly high res mipmaps

- #6192 Crash when removing and recreating a procedural instance
- #6202 counter overflow crash in big scanline EXR images
- #6203 Camera corruption after substantial forward zooming with kick -ipr m
- #6219 photometric\_light filename does not support environment variable expansion
- #6222 Metallic BSDF albedo is always white
- #6242 Crash when removing a procedural node
- #6245 shadow\_matte AOVs
- #6248 Remove offscreen\_color from shadow\_matte
- #6249 Remove shadow\_matte background\_type
- #6250 Re-introduce background\_color in shadow\_matte
- #6251 shadow\_matte self-reflections / self-shadowing
- #6254 trace\_set: crash when destroying shader
- #6255 sss\_irradiance\_shader fails with closure-based shaders
- #6256 Add alpha\_mask in shadow\_matte
- #6286 Volume shader: crash when connected as atmosphere shader
- #6287 Crash with invalid options.atmosphere and options.background shaders
- #6288 Miscellaneous crashes (empty nurbs, empty implicit, atmosphere shadow\_matte)
- #6292 Deep Driver: write errors hang the render
- #6293 Deep driver: append does not work with overscan renders
- #6294 Bucket call back: user bucket coords should be snapped to bucket grid
- #6295 Crash when saving empty ginstance with open\_procs enabled
- #6297 Camera differential evaluation can cause crashes or hangs with OSL shaders linked to camera
- #6313 crash when 4-channel opacity texture has no alpha=0
- #6331 Crash on Windows when loading plugins without .dll extension
- #6353 barndoors light filter result not order-independent
- #6174 "host app" metadata item not readable by AiMetadataStoreLoadFromASS
- #6258 Range shader can sometimes output infinite or nan
- #6303 Support upper and mixed case versions of .ass / .ass.gz extensions

## 5.0.1.5

### Bug fixes

- #6242 Crash when removing a procedural node
- #6254 trace\_set: crash when destroying shader
- #6255 sss\_irradiance\_shader fails with closure-based shaders
- #6286 Volume shader: crash when connected as atmosphere shader
- #6288 Miscellaneous crashes (empty nurbs, empty implicit, atmosphere shadow\_matte)
- #6292 Deep Driver: write errors hang the render
- #6293 Deep driver: append does not work with overscan renders
- #6294 Bucket call back: user bucket coords should be snapped to bucket grid
- #6295 Crash when saving empty ginstance with open\_procs enabled
- #6297 Camera differential evaluation can cause crashes or hangs with OSL shaders linked to camera
- #6258 Range shader can sometimes output infinite or nan

## 5.0.1.4

### Bug fixes

- #6177 wireframe artifacts when not in raster-space
- #6184 MtoA crash when saving a scene using motion blur
- #6186 triplanar uses wrong mipmap level for some parts of the object
- #6187 triplanar uses overly high res mipmaps
- #6192 Crash when removing and recreating a procedural instance
- #6202 counter overflow crash in big scanline EXR images
- #6203 Camera corruption after substantial forward zooming with kick -ipr m
- #6219 photometric\_light filename does not support environment variable expansion
- #6222 Metallic BSDF albedo is always white
- #6174 "host app" metadata item not readable by AiMetadataStoreLoadFromASS

## 5.0.1.3

### Bug fixes

- #3319 Alpha not fully opaque in output images
- #6104 Quad lights do not work with projected textures
- #6136 Indirect sample clamp not visible in AOVs
- #6151 Black AOV output due to conflicting AOV type redefinition
- #6162 AiShaderGlobalsGetPositionAtTime, AiTraceBackground, and AiVolumeSample crash in background shading context
- #6166 AiM4Scaling does not work in python
- #6168 Overriding view direction for metal BSDF not supported
- #6170 Crash creating motion blurred min\_pixel\_width inside a procedural

## 5.0.1.2

### Bug fixes

- #6128 curvature shading differences when seen through glossy transmission/reflection
- #6135 crash during accel construction if there are over 65k overlapping primitives
- #6143 AiShaderGlobalsGetVertexUVs not working for uvsets in free render mode

## 5.0.1.1

### Bug fixes

- #5981 Incorrect melanin absorption values for wide gamut rendering color spaces
- #6089 Deep EXR output of light path expressions missing volumes
- #6094 Artifacts in lat long skydome\_light
- #6095 Fresnel discontinuity in diffuse term when texture is connected to specular\_color
- #6097 Subdiv: duplicate vertex index in face causes crash
- #6116 Warn that images will be watermarked if license authorization fails
- #6124 Mesh light crashing when provided non-mesh node

## 5.0.1.0

### Enhancements

- **Improved mesh light sampling:** A new sampling algorithm has been implemented for mesh lights which can greatly improve their rendering quality when shading points near the light's surface, particularly in volumes. (#6005, #6062, #6074)
- **Indirect sample clamp:** The new `options.indirect_sample_clamp` parameter works similar to the existing `AA_sample_clamp`, but only affects indirect light, so that specular highlights from direct lighting are preserved. This makes it possible to clamp fireflies more aggressively without affecting the final render significantly. It is set to 10.0 by default. Lower values result in more aggressive noise reduction, possibly at the expense of dynamic range. (#5978)
- **Sharper quad and disk light spread:** Due to a more mathematically robust implementation, quad and disk lights now allow for more focused beams when their `spread` parameter is at zero. (#5980)
- **Faster BVH builds:** The BVH ray accel data structures are now faster to build, specially on machines with many cores. On a dual-socket 56-thread machine we've seen up to 2x faster builds and on a quad-socket 64-thread machine it's up to 3x faster. (#3474, #5951, #5962, #6001)
- **Polymesh as volume container:** The `polymesh` node now can be a volume container when `step_size` is greater than zero, and in that mode it will interpret the assigned shader as a volume shader. This is similar to other volume containers like `sphere` and `box`. A built-in channel named "density" is provided for shaders to query, which will be 1.0 on the inside of the polymesh, and 0.0 outside. To facilitate positional displacement in the volume shader, which is commonly used to add detail such as noise, there is an additional parameter `volume_padding` that will provide extra room. Note that large volume padding values, particularly when combined with high polygon counts (including through subdivision) will hurt performance, so only use the amount of subdivision and padding you need. (#6039, #6057, #6059, #6060)
- **Padding in volume-capable primitives:** In addition to the `polymesh` node, now `points`, `box` and `sphere` when in volume mode can have `volume_padding` applied so that positional displacement can be added in the volume shader. (#6055, #6059)
- **Iridescence in standard\_surface:** Iridescence effects can be controlled by the new parameters `thin_film_thickness` and `thin_film_IOR` in the `standard_surface` shader, affecting the specular, transmission and coat components. This uses a new per-microfacet Airy formula, supports artistic Fresnel or complex IOR, and gives much more accurate results than the older `thin_film` shader. Typical thickness values are in the 0nm to 2000nm range. The effect will disappear above 3000nm. (#5750)
- **Exact Fresnel equation:** The exact Fresnel equation is now used for both microfacet BRDF and microfacet refraction BSDF instead of Schlick's approximation. (#5371)
- **Coat AOV:** New built-in AOVs were added for the `standard_surface` shader: `coat`, `coat_direct`, `coat_indirect` and `coat_albedo` AOVs. The builtin `specular` AOVs now only contain the base specular layer. (#6014)
- **Skydome Light AOV:** The skydome light is now output to direct light AOVs by default again as in Arnold 4. It can still be output to the indirect light AOV by enabling the `skydome_light.aov_indirect` parameter. (#6022)
- **Global AOV shaders:** A new list of shaders can be defined in `options.aov_shaders` that will be evaluated after the regular surface shader. With this it's possible to add shaders to set specific AOVs without modifying the original shader tree. Shaders intended for this purpose should add a boolean metadata named `aov_shader` on the node itself, as a user-interface hint. If `options.atmosphere` or `options.background` are set, these global AOV shaders will also be run for atmosphere and background contexts. (#5942)
- **Normal map shader:** A strength parameter was added to increase or decrease the effect of the `normal_map` shader, with the default 1.0 applying the normal map unmodified. The `normalize` parameter was removed and the output normal is now always normalized. (#5971)
- **Camera barrel distortion:** A new `persp_camera.radial_distortion` attribute has been added to specify the camera's quadratic radial distortion, with negative and positive values resulting in pincushion and barrel distortion respectively. (#2385)
- **Camera FOV EXR metadata:** As a convenience, a new field `CameraFov` has been added to EXR metadata to explicitly store the render camera's FOV. This is a lot easier than having to derive it from the film aperture and focal length with the formula `fov = 2 * Rad_To_Deg * atan(CameraFilmApertureHorizontal / (2*CameraFocalLength))`. (#6026)
- **Half-float EXR layers:** Output layers in files that support layers (such as regular or deep EXR files) can be individually set to type `HALF` by adding an optional `HALF` modifier to the corresponding output string. For instance: "my\_aov RGB filter driver HALF". (#3839)
- **Custom EXR layer names:** Output layer names can be customized in file formats that support layers (regular or deep EXR) by adding an optional custom layer name after the driver name in the `options.outputs` string: (#6047)

```
outputs 2 1 STRING
"RGBA RGBA gaussian_filter driver_exr beauty"
"RGBA RGBA variance_filter driver_exr beauty_variance"
```

- **OCIO display / view:** OCIO Display/View tuples can now be selected as an output color space following Maya's convention. This applies to any input or driver node with a `color_space` string attribute. When using the OCIO color manager, if a color space name matches the format "`VIEW_NAME (DISPLAY_NAME)`" the given view on the given display will be used if they both exist. Note that conversions to a given display/view might not be valid for input textures, but they will always be available for output drivers. (#6069)
- **Weighted variance filter:** The variance filter has a new `variance_filter.filter_weights` to set the weights used for each sample. The default value (`box`) matches the previous behavior. (#5954)

- **Array setting with kick:** Array parameters of type `float` can now be set in the command line with `kick -set`. For example, `kic k -set mynode.myarray 0.1 0.2 0.3 0.4` will create and assign a 4-valued array to `myarray`. (#6043)
- **Upgraded OIIO to 1.7.15:** We have upgraded to OIIO 1.7.15 which provides for several bug fixes and enhancements. Improvements not listed in our bug fixes section are (#6009):
  - PSD files now have support for cmyk, multichannel, and grayscale color modes and now supports 32 bit per sample bit depths.
  - Log statistics no longer list as "BROKEN" files which were invalidated or that were initialized but never actually opened.
- **Upgraded OSL to 1.9.0:** Open Shading Language has been upgraded to the latest version. This brings many performance improvements and bugfixes, along with new features such as operator overloading, struct initializers, and new utility functions. (#6034)

## API additions

- **ASS file metadata:** New API functions are provided to store and read metadata from .ass scene files. Metadata is added as special comments at the beginning of the file. For now, all metadata types are stored and read as strings. (#3833)
- **Non-destructive AiASSWrite():** It is no longer necessary to enable `options.preserve_scene_data` before writing a scene to an .ass file in order to be able to render it or write it again afterwards. (#3182)
- **AiMakeTx:** `AiMakeTx()` can now be run with an OCIO config, even outside an `AiBegin()` / `AiEnd()` block. Just specify `--colorconfig <path_to_ocio_config>` as an additional flag. All concurrent jobs must use the same config and convert to the same linear rendering color space. (#5786)
- **AiShaderGlobalsGetShader():** Return root shader node assigned to the object at the current shading point. (#5956)
- **AiShaderGlobals():** This function now allocates a more complete shader globals, so that when used with `AiTraceProbe()` user data is accessible. (#5956).
- **AtHPoint C++ math operators:** In addition to `AiV4Add()`, and associated math functions, operators +, -, \*, and negation have been added in order to allow for cleaner code. A `project()` member function has also been added which can be used instead of `AiV4Project()`. The `AiV4CreateVector()` and `AiV4CreatePoint()` functions have now been overloaded so that the following is also possible:  
`AtHPoint hp = AiV4CreatePoint(my_AtVector);` (#5918)
- **AiMicrofacetSetThinFilm():** This can be used to add thin film effects to the microfacet, metal and refraction BSDFs. It works by replacing the classic Fresnel term with a new Airy reflectance term, giving more accurate results than the `thin_film` shader. The microfacet normal is used instead of `sg->N`. (#5750)
- **AiVolumeMergeIntersection():** Like `AiVolumeAddIntersection()`, this is used by a volume or implicit plugin to submit a ray interval for integration or intersection. However, it will merge with other extents submitted that match, so that any overlapping intervals are not integrated and shaded more than once. Previously a plugin would have to do this merging itself. (#6058)

## Incompatible changes

- **Indirect sample clamp:** This feature is enabled by default (at 10) and in most scenes will have little impact besides reducing noise. For fully backwards compatible renders this parameter may be set to a very high value like `1e30`. (#5978)
- **Exact Fresnel equation:** Due to the change of the Fresnel term, there will be differences especially when the relative IOR is close to 1, for example, when light travels from water to ice. (#5371)
- **OSL texture color space:** By default no automatic color space conversion is applied anymore in `texture()` calls. To revert to the previous behavior, use the optional `colorspace` argument set to `auto`: `texture("filename.jpg", u, v, "colorspace", "auto")`. (#6030)
- **bounds\_slack renamed to volume\_padding:** In volume nodes (those backed by plugins, such as the built-in OpenVDB volume node) the `bounds_slack` parameter has been renamed to `volume_padding` to be consistent with the other nodes. A synonym exists for `bounds_slack` so it will continue to work, but will emit a warning when used to remind you to switch to the new parameter name. (#6059)
- **max\_subdivisions:** The type of the global `options.max_subdivisions` has been changed from `AI_TYPE_INT` to `AI_TYPE_BYTE`, to match the type of `polymesh.subdiv_iterations`. Existing client code should set this global option with `AiNodeSetByType()` to avoid a (harmless) warning. (#6029)
- **light\_decay output type changed:** The shader output type of the `light_decay` light filter shader changed from `AI_TYPE_FLOAT` to `AI_TYPE_NONE`. It doesn't actually output anything, it just modifies the shader globals as do the other light filters. This makes its output type consistent with the other light filter shaders. Any shader networks that were relying on its output should be disconnected from it. (#6076)
- **OCIO config defaults:** For well known OCIO configs Arnold will use a name-based heuristic to determine reasonable default color spaces and chromaticities. These new defaults will be reported in the log file. This applies to ACES, nuke-default, spi and filmic-blender configs. Previously Rec. 709 chromaticities were always assumed. (#6035)

## Bug fixes

- #5927 Memory pools allocations are much bigger than requested
- #5639 Mesh lights in procedurals not working properly
- #5839 Metadata files don't allow boolean values as attribute or metadata name
- #5870 skydome\_light and quad\_light incorrectly detect some textures as constant color
- #5880 camera\_projection : use NODE instead of STRING
- #5910 Missing Python bindings for some AtNodeEntry API functions
- #5911 Nodes contained in .obj and .ply procedurals are registered in the global name scope
- #5917 Remove pykick since it's broken
- #5923 UINT parameter values are clamped to 0x7FFFFFFF when parsed from .ass files
- #5925 Properly handle cases where cpuset is smaller than the detected number of cores
- #5926 ignore\_motion\_blur with non-zero reference\_time does not work with a polymesh with normals
- #5930 normal\_map shader issues
- #5931 Standard surface coat normal not decoupled from main normal
- #5936 AiNodeClone not working for parameter overrides
- #5940 AiNodeEntryGetDerivedType wrong for builtin procedural and implicit nodes
- #5946 Inconsistent reference\_time between deform and transform motion blur
- #5947 AiSetAppString() not working
- #5950 Improve precision of box and plane objects
- #5953 Wrong stats after destroying shape nodes
- #5964 LPEs not working with raw drivers
- #5969 Avoid gcc -Wall warning about strict-aliasing in AtParamValue
- #5973 Curves with varying UVs rendering wrong
- #5977 Shadow matte shader missing transparency with passthrough shader
- #5982 Memory leak due to AiShaderEvalParamArray
- #5988 Avoid gcc -Wall warning about strict-aliasing in AtArray
- #5991 OpenEXR metadata not working for 3D vectors and RGB colors
- #5992 Color Manager: support chromaticities for additional standard gamuts / white points
- #5994 Color manager: wrong chromaticities for XYZ rendering color space
- #6001 Performance regression in BVH build
- #6015 Crash with oriented curves mode when no orientations are specified
- #6028 Rendering color space: use more accurate conversion matrix
- #6029 tighten max\_subdivisions from 999 to 255
- #6030 Set OSL texture() default color transform to no-op
- #6031 OSL refraction closure crash
- #6036 Missing nodes not aborting render
- #6041 Color Manager: account for white reference when chromaticities are specified
- #6044 Color Manager: test and warn for degenerate rendering color space chromaticities
- #6045 Color management: review warning and error severity
- #6049 AtTextureHandle becomes corrupt after invalidating corresponding opacity texture
- #6053 AiNoise\*() returns black when number of octaves is large
- #6076 Change light\_decay output type to AI\_TYPE\_NONE
- #5899 Missing linkable metadata for some shader parameters
- #6080 trace() in OSL doesn't give the closest hit point

## 5.0.0.3

### Bug fixes

- #5639 Mesh lights in procedurals not working properly
- #5982 Memory leak due to AiShaderEvalParamArray
- #5988 Avoid gcc -Wall warning about strict-aliasing in AtArray
- #5992 Color Manager: support chromaticities for additional standard gamuts / white points
- #6001 Performance regression in BVH build
- #6015 Crash with oriented curves mode when no orientations are specified

## 5.0.0.2

### Bug fixes

- #5946 Inconsistent reference\_time between deform and transform motion blur
- #5947 AiSetAppString() not working
- #5953 Wrong stats after destroying shape nodes
- #5964 LPEs not working with raw drivers
- #5969 Avoid gcc -Wall warning about strict-aliasing in AtParamValue
- #5973 Curves with varying UVs rendering wrong
- #5977 Shadow matte shader missing transparency with passthrough shader

## 5.0.0.1

### Bug fixes

- #5927 Memory pools allocations are much bigger than requested
- #5910 Missing Python bindings for some AtNodeEntry API functions
- #5911 Nodes contained in .obj and .ply procedurals are registered in the global name scope
- #5917 Remove pykick since it's broken
- #5923 UINT parameter values are clamped to 0x7FFFFFFF when parsed from .ass files
- #5925 Properly handle cases where cpuset is smaller than the detected number of cores
- #5926 ignore\_motion\_blur with non-zero reference\_time does not work with a polymesh with normals
- #5930 normal\_map shader issues
- #5931 Standard surface coat normal not decoupled from main normal
- #5936 AiNodeClone not working for parameter overrides
- #5940 AiNodeEntryGetDerivedType wrong for builtin procedural and implicit nodes

# 5.0.0

## Enhancements

### Shaders

- **Standard Surface:** a new `standard_surface` shader was added, with intuitive and energy conserving parameters, a secondary specular coat with separate normal, metallic Fresnel, thin surface support and more. The `standard` shader is still available but considered deprecated. (#5372)
- **Standard Hair:** a new physically based `standard_hair` shader was added, with much more accurate specular and transmissive scattering, better diffuse scattering for dirty hair, melanin randomization, and simple and intuitive parameters. The older hair shader is still available but considered deprecated. (#5370, #5851)
- **Standard Volume:** a new `standard_volume` shader was added, usable for rendering a wide variety of volumes. The shader provides independent control of density, scattering color and transparency, in a way that is energy conserving by default. Fire can be rendered using blackbody emission driven by temperature. Displacement can be used to add more detail to volumes. This supersedes the density volume shader, which has been removed. (#5090)
- **Shader mixing:** surface, hair and volume shaders now output closures rather than colors. This makes it possible to mix shaders efficiently, using the new `mix_shader` node to blend or add two shaders including all their light AOVs. Adopted from the common shaders, there is also a vanilla color mixing shader `mix_rgba`. The ray switch shader has been split up into two `ray_switch_shader` and `ray_switch_rgba` nodes, for switching between shaders and one for switching between texture colors respectively. (#5494, #5495)
- **Bump shaders:** `bump2d` and `bump3d` now output a normal vector that can be linked to new `normal` parameters in `ambocc`, `lambert`, `standard_surface` and `utility` shaders. These bump shaders no longer function as passthrough shaders. The `@before $` syntax for bump shaders has been removed as well. (#5599)
- **Noise shader:** The `noise.coord_space` parameter now has a `uv` enum value for texturing using the object's local UV coordinates. Note that this calls the faster 2D noise API, not the 3D noise like all other coordinate spaces. In addition, an arbitrary coordinate space can be specified manually by linking another shader into the new `P` parameter. The shader can now output colors, in scalar mode by blending between `color1` and `color2`, and in vector mode with a separate noise signal per color channel. A new `time` parameter can be used to smoothly vary noise over time. (#5195, #5235, #5247)
- **Utility shader:** the `utility` shader has been enhanced with a new `metal` shading mode, and a `roughness` control has been added that affects plastic and metal modes. (#5018)
- **Legacy standard shader:** the Beckmann specular distribution in the (deprecated) `standard` shader now uses more correct per-microfacet fresnel, as was already used for GGX. `sss_profile` has been removed and the empirical BSSRDF is now always used. (#3285)
- **Legacy hair shader:** `uparam` and `vparam` have been removed from the (deprecated) `hair` shader. Instead, curves now support UV sets. This shader does not fully support light path expressions. (#5646)
- **Env vars in image filename:** the `image` node `filename` field will now expand environment variables of the form `[var]`, in addition to it already expanding them when they are in the texture searchpath. (#5671)
- **New tags in image filename:** the `image` node `filename` field now accepts `<utile>` and `<vtile>` tags. They take an optional integer offset `<utile:2>` that adds to the tile number; e.g. an offset of 2 from a U coordinate of 1.4 will output tile number 3. (#5753)
- **Renamed image missing tile parameters:** on the `image` texturing node, `ignore_missing_tiles` has been renamed to `ignore_missing_textures`, and `missing_tile_color` has been renamed to `missing_texture_color`. The handling of missing textures has been extended to non-tagged textures. It is still recommended to not ignore missing textures and fix them (the render will emit an error to the log and abort if `options.abort_on_error` is left on). However in some cases missing UDIM or other tiled textures may be a deliberate choice. These options allow control, per image node, what happens in those cases. (#5753)
- **Built-in utility common shaders:** The common utility shaders that shipped at least in part with MtoA, C4DtoA, HtoA and KtoA have been integrated into the core. A few shaders from the set have not been added, such as `print`, `linearize`, `ln` and the matrix shaders, and a few have improved parameter defaults, and others have been optimized, but they are largely the same. (#5714, #5749)
- **Color Jittering shader:** a new `color_jitter` shader can be used to generate random colors within a specified gain, hue and saturation range. Colors may be randomized per face, object, procedural instance or user data, or a combination. (#5793)
- **Triplanar Shader:** a new `triplanar` shader can be used to quickly map image textures without needing UV coordinates. The texture is projected onto the object from the 6 sides, and smoothly blended together at the seams. (#5770)
- **Dedicated normal and vector map shaders:** The common shader `space_transform` had extra functionality for dealing with tangent-space normal or vector maps, and that has been removed. Instead, there are now two new shaders, `normal_map` and `vector_map` that take typical normal map and vector displacement map data, respectively, and prep them for their typical use later in a shading network, similar to how the bump shaders would be used. (#5714)
- **os1 shader:** while it is recommended to compile `.osl` files to `.oso` and place them in the shader path, OSL shader code can be referred to directly via the `os1` node or even set as a string parameter. The shader node will then generate all the parameters of the OSL code with the `param_` prefix in front of each. (#5471)
- **Skydome light camera visibility:** New `camera` and `transmission` parameters set the amount of light contributed to camera and specular transmission rays, so that it is no longer required to use a separate background shader for such purposes. A new shader `p`

arameter may be used to link closure shaders to control color and transparency. Skydome lights are now preferred over background shaders, as they provide the same functionality with better sampling. (#5744)

## Sampling

- **Dithered sampling:** most samplers (e.g. soft shadows, indirect illumination, depth of field) will now take advantage of dithered sampling, which improves the visual distribution of noise specially at low sample rates. (#5047, #5412)
- **Quad lights:** sampling has been improved, reducing noise for surface lighting. For comparisons with the previous sampler, option `s.enable_new_quad_light_sampler` can be disabled. (#4961, #5780)
- **Cylinder lights:** sampling has been improved, significantly reducing noise for cylinder lights in volumes or where cylinder lights are located near other surfaces. (#5347)
- **Disk lights:** a novel sampling algorithm has been implemented for disk lights which can greatly improve their rendering quality when shading points near the disk's surface, particularly in volumes. (#5316, #5856, #5862)
- **Russian roulette:** on average better performance for hair, transmission and volume scattering. (#4052, #5789, #5690)
- **Motion blur time samples:** rendering motion blur is now faster when using non-default `deform_time_samples` or `transform_time_samples`. (#5336)
- **Caustic noise reduction:** a new method was added to reduce noise from caustics. Noise from GGX microfacet speculars in particular is reduced. `options.indirect_specular_blur` controls the trade-off between more accurate noisy renders at 0.0, and more blurry biased renders with reduced noise at higher values. (#4498)
- **Faster opacity mapping:** opacity-mapped transparent surfaces, such as tree leaves, are now sampled more efficiently and can render up to 20% faster, specially in machines with many threads. (#4704)

## OSL, Closures and AOVs

- **Open Shading Language:** shaders can now be written in Open Shading Language, an advanced shading language for production GI renderers. OSL shaders placed in the shader search path are automatically registered as Arnold shader nodes, with their parameters converted to Arnold parameters. Once loaded, they can be inspected, instantiated and linked in exactly the same way as C++ shaders. OSL shaders can be used to implement texture patterns and materials using closures. See the [OSL documentation](#) for more details. (#4357, #5400, #5487, #5526, #5471)
- **Closures:** a new closure parameter type has been added, which shaders can output instead of final colors. There are BSDF, BSSRDF, emission, matte, transparency and volume closures. See the API documentation and [examples](#) for more details on how to use these. Linking a color to a closure parameter will automatically create an emission closure with that color. A closure parameter however can't be linked or converted to a color, as the integrator only computes lighting after shader evaluation. (#4793)
- **Light group AOVs:** surface shaders now natively support light group AOVs, previously this feature was only available for volume shading. (#4305)
- **Light path expressions:** light path expressions are used to write lighting components into separate AOVs. No longer should individual shaders define AOVs for direct/indirect light and various layers, rather a regular expression syntax is used to define the subset of all scattering and emission events in the scene that should be written to each AOV. Built-in AOVs are available for the common cases. See the [LPE documentation](#) for details. (#5491, #5581, #5582)

## Color Management

- **Color Managers:** custom `color_manager` nodes can now be implemented to handle input and output color transforms. A color manager is a package similar to `SynColor` or `OpenColorIO`. If no color manager is specified in `options.color_manager`, Arnold will use the built-in one which supports `linear` (no transform), `sRGB` and `Rec709` (non linear) (#5262, #5527)
- **OpenColorIO:** color spaces defined in `OpenColorIO` configurations can now be used in `.ass` files through the built-in `color_manager_ocio` node. (#5598)
- **Linear color spaces:** any linear color space can now be specified through the color manager for any output driver that supports halves or floats. (#5552)
- **Color space metadata:** OpenEXR files have additional metadata fields with color manager type (`arnold/color_manager`), color space name (`arnold/color_space`) and chromaticities (`ColorSpace/Red Primary`, etc) when known. (#5263)
- **Rendering color space:** using specific implementations of a color manager it is now possible to specify Arnold's rendering color space. Arnold's default rendering color space is `Linear sRGB`. For other color spaces Arnold will try to auto-detect chromaticities and illuminant, and will expect all data (textures and `.ass` file parameters) in that same color space. An important note is that different rendering color spaces will yield different render results when compared with the default `sRGB linear`. These color and intensity shifts will be more visible with transmission, but will also affect illumination and subsurface. Because of this, shader adjustments and texturing should happen in the same color space as the rendering one. (#5262, #5527, #5819)
- **Color space-independent spectral effects:** effects that perform spectral integration, like refractive dispersion, `thin_film`, `black_body` and `physical_sky` now take into account the rendering color space and provide accurate results for wider gamut spaces, or spaces with different illuminants. There might be some differences for these effects, like higher saturation, due to improved color computations. (#5819, #5835, #5836, #5837)
- **Listing available color spaces:** `kick -lcs` prints a list of available color spaces. (#5832)

## Namespaces

- **Namespaces:** Arnold scenes are organized in hierarchies of procedurals, which previously could lead to naming conflicts when identically names nodes existed in different procedurals. Each procedural now has its own naming scope, with the following rules: (#2712)
  - In procedurals .ass files, node names are looked up first in the current scope. If not found, they are looked up in the parent scopes until the scene root is reached.
  - Alternatively, an absolute path may be specified to look up a node starting from the scene root, for example: "`^root_node ^proc1^proc2^node_name`".
  - `AiNodeLookUpByName()` now has a `parent` parameter to specify the scope where recursive lookup begins.
  - Procedural plugins creating nodes using `AiNode()` and `AiNodeClone()` must now set the `parent` parameter, to indicate that the node is a child of the current procedural node.
- **Procedural Node Overrides:** nodes inside a procedural can be replaced by other nodes with the `override_nodes` parameter. This may be used for example to replace shaders in an existing .ass procedural. When the parameter is enabled, nodes in the immediate parent scope of the procedural will replace identically named nodes inside the procedural. (#2712)

## Other Enhancements

- **OpenVDB:** rendering of .vdb files is now supported in the Arnold core, using OpenVDB 4.0. The `volume` node now has a `filename` parameter that accepts .vdb files. The `volume_implicit` node may be used to render OpenVDB volumes as an implicit surface. The parameters for both nodes are the same as in the previous OpenVDB plugin, except for the velocity shutter start and end which has been replaced by `motion_start` and `motion_end` parameters shared with other shapes. Velocity grids named `v`, `vel` or `velocity` are automatically detected and used if motion blur is enabled in the scene. `options.texture_searchpath` is used to resolve relative filepaths. (#4844, #5801, #5814)
- **Improved OpenVDB startup time:** On Linux when rendering with many threads, large OpenVDB datasets could cause significant slowdowns at the start of rendering. Startup times are noticeably improved, resulting in buckets rendering faster especially for low AA samples such as during IPR. (#5816)
- **Faster implicit surfaces:** Implicit surfaces, through the `implicit` geometric primitive, render faster, particularly when using the uniform solver. The uniform solver requires fewer solver samples to accomplish much-improved quality, and the levelset solver got slightly faster as well. (#3221, #5472, #2097)
- **Faster curves:** The `curves` geometric primitive now renders about 5 to 15% faster. In addition, when rendering dense hair clumps, `min_pixel_width` is now up to 2x faster and results in more accurate shadowing, at the expense of a slight increase in sampling noise. (#4417, #5725, #5806, #5684, #5806)
- **Faster object initialization:** The initialization time for motion blurred objects is now about 700ns faster. For a scene with 1M motion blurred objects, that's a savings of about 0.7s. (#5379)
- **VR camera:** a new `vr_camera` node is now available that generates stereo views suitable for virtual reality in a variety of formats and projections. (#4906)
- **Quad and spot light roundness:** Quad and spot lights now have a `roundness` parameter, going from a square shape at 0, to rounded corners, to a disk shape at 1. Quad lights also have a new `soft_edge` parameter to soften the edges, similar to the `penumbra_angle` for spot lights. (#5144, #5222)
- **Lazy evaluation of shader array linking:** When calling `AiShaderEvalParamArray()`, no shaders are evaluated anymore. Reading one of the array values later on will trigger the corresponding shader evaluation, and the result is cached to avoid multiple shader evaluations. This means performance no longer depends on the number of links, which can be scaled as necessary. (#2924)
- **RGB camera filter maps:** The data type of camera filter maps has changed from float to `AtRGB` so that colored effects are possible. (#5299)
- **Object transform\_type:** Shape objects (`ginstance`, `polymesh`, etc.) now have a `transform_type` parameter that specifies what type of motion the object has. Options are `linear`, `rotate_about_center`, and `rotate_about_origin`. `linear` corresponds to the linear interpolation between matrices that Arnold 4.2 used to do when `options.curved_motionblur=false`. `rotate_about_origin` corresponds to `curved_motionblur=true`. Unlike `rotate_about_origin`, which sets the rotation pivot at the origin, `rotate_about_center` will rotate about the object's center. This is the default mode and is useful for wheels, propellers, and other objects which spin. (#3962, #5376)
- **Hair UVs:** `curves.uvs` is a new parameter to set `sg->u` and `sg->v` default texture coordinates, similar to polymeshes. UVs may be specified per curve or per point, with an array that has the same length as uniform and varying user data respectively. `sg->bu` and `sg->bv` still contain the UV coordinates along the width and length of the curve respectively. The hair parameters `uparam` and `vparam` have been removed, as they can now be set through `curves.uvs`. (#5646)
- **ARNOLD\_PLUGIN\_PATH:** This environment variable is now used automatically in `AiBegin()` to load plugins, and to find procedural and volume DSOs. Previously this was a convention used by kick and the DCC plugins. (#5282)
- **kick camera controls:** New command-line flags to tweak the current render camera have been added. You can now set the position (-position), up vector (-up) and look at point (-lookat) for the current camera. This means that `kick -turn` now works for cameras defined with a matrix if you use the `-lookat` flag. (#5311)
- **kick exposure keys:** The [ and ] keys can be used to decrease or increase exposure in `kick` when in `-ipr` mode. (#5380)
- **kick abort on error:** `kick` now aborts on errors during loading of .ass files by default, to avoid rendering when plugins are missing or files are corrupted. Previously it would only abort on errors during rendering. Aborting can still be disabled with `-set options.abort_on_error off`. (#5736)
- **Polygon holes:** a new attribute `polymesh.polygon_holes` is available to add holes to given faces of a mesh. For instance,

```

polymesh
{
    name quad_with_hole
    nsides 2 1 UINT 4 4
    polygon_holes 2 1 UINT 1 0
    ...

```

indicates that the face with index 1 will be a polygon hole on face 0. Multiple holes per face are supported. The winding of a face used as a hole does not need to be reversed. Also, note that polygonal holes are ignored when a mesh is subdivided. (#2972)

- **Tagging deprecated nodes:** Nodes can be tagged as deprecated by adding boolean metadata named "deprecated" (with the parameter name empty) set to true. These nodes will be listed in `kick -nodes with [deprecated]`, and when they are instantiated in an Arnold scene they will issue a warning to remind the user. (#5746)
- **Updated to RLM 12.2BL2:** We have upgraded the license server and the external library controlling the licensing subsystem from version 12.0BL2 to 12.2BL2, a more stable release fixing various crashes, bugs, hangs and memory leaks. (#5754)
- **Updated to OIIO 1.7.12:** We have upgraded from OIIO 1.7.7 to OIIO 1.7.12. (#5672, #5826)

## Incompatible changes

### Options

- Added `options.subdiv_dicing_camera`, replacing the per polymesh adaptive subdivision dicing camera with a global one. (#5029)
- Replaced `options.aspect_ratio` with `options.pixel_aspect_ratio`. The new pixel aspect ratio follows the usual definition `pixel_width / pixel_height`. Note that the new attribute is the inverse of the old `options.aspect_ratio`. (#5392)
- Renamed `options.shader_searchpath` to `options.plugin_searchpath`, all types of node plugins are loaded from this path. (#5779)
- Removed `enable_fast_opacity`, so that the fast opacity mode is always used. (#5158)
- Removed `auto_transparency_threshold`: The `options.auto_transparency_threshold` has been removed. As long as shaders make sure to use `AiShaderGlobalsStochasticOpacity()`, there is no need for manually specifying a transparency threshold. (#4842)
- Removed `ignore_direct_lighting`: The AOV system provides various indirect lighting AOVs that can achieve the same result as this old debugging option. (#5029)
- Removed `options.curved_motionblur`, in favor of a per object `transform_type` parameter. (#5437)
- Removed `options.bump_space` and `options.bump_multiplier`. Bump mapping is now always in object space. (#5029)
- Removed `options.GI_falloff_start_dist` and `options.GI_falloff_stop_dist` have been removed. (#4793)
- Removed `options.light_gamma` and `options.shader_gamma`. All shader and light attributes are now assumed linear. The `always_linear` metadata is therefore no longer needed and is no longer supported. (#5245)
- Removed rarely used `bottom`, `right` and `woven` modes from the `options.bucket_scanning` enum. (#5642)
- Removed `options.enable_threaded_procedurals`, this was previously already enabled by default. We now require clients to make their geometry procedurals thread safe. (#5029)
- Removed `options.CCW_points`. If polygons have a clockwise winding order, they are now expected to be converted to counter-clockwise when translating the geometry. (#5029)
- Removed `options.shadows_obey_light_linking`, this is now off as was already the default. To have light groups affect shadows, set the shadow group to the same value as the light group. (#5029)
- `options.preserve_scene_data` is no longer propagated when writing to an .ass file. (#5651)
- Removed `options.procedural_force_expand`, since procedurals are always expanded during initialization now. (#5612)

### Lights

- Lights with `normalize off` and radius zero now no longer emit any light, for consistency with lights with very small radii. Previously these would work as if `normalize` was on. (#3851)
- Light sampling now uses `options.GI_diffuse_samples` and `options.GI_specular_samples` for multiple importance sampling. If existing scenes render with more noise, increasing these samples maybe be necessary. (#5423)
- Removed constant light decay, and the `decay_type` parameter, so that all lights now have quadratic decay. (#5147)
- Removed support for negative lights (negative color, intensity, or shadow\_color). Negative values will now be clamped to zero. (#5162)
- Removed `affect_diffuse`, `affect_specular` and `affect_volumetrics` parameters. Instead `diffuse`, `specular`, `sss` and `d` volume multipliers can be set to zero. (#5423)
- Renamed `gobo` parameters `scale_s`, `scale_t`, `wrap_s`, `wrap_t` to `sscale`, `tscale`, `swrap` and `twrap`. (#5183)

### Ray Types

- The ray types used for ray visibility, the ray switch shader, GI depth and samples have changed. There are now 5 scattering ray types: diffuse reflection, diffuse transmission (includes SSS), specular reflection, specular transmission and volume scattering. In the options node there is now a GI depth and number of samples for diffuse, specular and transmission rays. `kick` now has arguments to set the diffuse, specular and transmission depth and samples. (#3462, #5572)

## Motion Blur

- Arnold and its DCC plugins used to encourage the use of "canonical" 0..1 time for all data in .ass files and in Arnold. For example, if motion blur samples went from frame-relative time of -0.25 to 0.25, in Arnold-land the -0.25 time sample would become 0.0 and the 0.25 time sample would become 1.0. This simplified the data in .ass files, etc but made it difficult or impossible to share .ass standins among vendors or even shots that differed in their time ranges, and to know why the motion blur was mismatched. Now, the `transform_time_samples` and `deform_time_samples` arrays, along with `time_samples` arrays on lights and cameras have all been removed in favor of the simpler `motion_start` and `motion_end` float parameters on each object. These should always be frame-relative values, e.g. -0.25 and 0.25 respectively in the example above. *NOTE* that this means non-uniform motion time samples are no longer possible; if an object has N motion keys, the time interval will always be divided up into N-1 equally-space time intervals. Also, this means that transform motion blur and deformation motion blur will always use the same time range. (#4916)
- Volume motion blur is also controlled by `motion_start` and `motion_end` parameters. The `velocity_fps` should be set to the framerate, so that the velocity can be scaled to the length of the frame. With the frame relative convention, motion start and end should typically default to -0.5 and 0.5 respectively, indicating the velocity was sampled at the center of the frame. (#4844)

## Other Changes

- Visual Studio 2015 redistributable:** In Windows, we now require the VS 2015 redistributable to be installed in order to run Arnold. We expect most of our users to have it already, but if not the installer can be downloaded from the following link: <https://www.microsoft.com/en-us/download/details.aspx?id=48145> (#4445)
- Geometry ID:** The type of geometry `id` attribute and of the associated built-in AOV has been changed from `INT` to `UINT`. (#5315)
- AOV filtering:** Built-in linear filters (`gaussian_filter`, etc) will no longer work on AOVs of type `INT`, `UINT` or `BOOL`. `closest_filter` should be used instead. (#5175)
- Custom cameras:** Custom cameras can now implement `camera_reverse_ray` to convert from camera space to raster space. This makes it possible for any custom camera to work as a dicing camera, support `min_pixel_width`, support accurate raster-space wireframe width, etc. (#4477)
- procedural node:** The procedural node now only supports loading .ass, .obj, .ply files, and creating procedurals using methods provided through `funcptr`. The `dso` parameter has been renamed to `filename`. `options.procedural_searchpath` is now only used for this node. Procedural plugins instead register new node types now. (#5154)
- Deferred procedurals:** Removed support for deferred procedurals. All procedurals are now loaded during scene initialization (right before rendering). Also, `load_at_init`, `min` and `max` parameters have been removed. (#5612)
- Gamma options removed:**
  - Output drivers now take a `color_space` string attribute to specify which color space to use for rendered images. Built-in values are `linear`, `sRGB` and `Rec709`. The default value `auto` will use `sRGB` for 8 bit formats and `linear` otherwise. All `driver_*`.`gamma` have been removed. (#5244)
  - image nodes now have an `color_space` attribute to specify which color space the texture is assumed to be in. Built-in values are `linear`, `sRGB` and `Rec709`. The default value `auto` will use `sRGB` for integer (8 or 16 bit) formats and `linear` otherwise. The attribute `options.texture_gamma` has been removed. (#5254)
  - Legacy gamma related options in `kick` (-g/tg/sg/lg) have been removed. You can now set the output color space for a render with `kick -ocs <string>`. (#5267)
- Removed -log legacy option in kick:** Use the more explicit `-logfile <path>` instead. (#5783)
- 32-bit Integer TIFF:** Removed the broken `int32` format from `driver_tiff`. Note that 32-bit integer TIFFs cannot be read by Nuke nor Gimp, and OSX Preview displays a reduced bit depth version, making this format pretty useless anyway. (#2976)
- Removed driver\_display:** The `driver_display` node, which was rarely used, has been removed. (#5270)
- Removed legacy pixel filters:** The following pixel filters, which were rarely used, have been removed: `catrom2d_filter`, `cook_filter`, `cone_filter`, `cubic_filter`, `disk_filter`, `video_filter`. (#5673)
- Always use auto-bounds for volumes and implicits:** Volume and implicit plugins already notify Arnold of the bounds around the data they provide. Users could previously override this with `min` and `max` parameters, but those have been removed for volume nodes and are unused for implicit plugins. All volume plugins will automatically have a `bounds_slack` parameter available to expand the natural bounding box around the data for purposes of adding positional noise in shaders, etc. Volume plugins are responsible for increasing their reported bounds by the `bounds_slack` amount. Also, the `load_at_init` parameter has been removed from both volume and implicit type nodes, as it had no effect in practice. (#5831)

## API changes

### General

- Many API functions now accept arguments by reference instead of by pointer. (#2159)
- Many API functions that used to have `char*` arguments or returns will now instead use `AtString`. Some of these changes should be transparent on account of the automatic conversion of `AtString` to `char*`, but in some situations, such as when the returned `AtString` is used by `AiMsg*`() or `printf` type variadic functions, preexisting code will not compile. In this case, the resulting `AtString` should have `.c_str()` called on it. (#5177)
- Added `AiMsgAddCallback()` to install an additional message handling callback alongside the default message processing. (#5736)
- Removed functions of the form `fooAtString(AtString str)` in favor of the overloaded `foo(AtString str)` forms. For instance, `AiNodeAtString(const AtString name)` is now `AiNode(const AtString name)` (#5177)
- Removed `AiLicense{Set|Get}Attempts()` and `AiLicense{Set|Get}AttemptDelay()`. (#2893)
- `AiMakeRay()` and other shader utilities now return their result. (#5300)
- Changes to support hierarchical namespaces:
  - `AiNode`: It now receives, in addition to the node entry name, the node name, and the parent (or `NULL` for a root node).
  - `AiNodeLookUpByName`: It receives a pointer to the parent node, so it can start relative search in the appropriate context (not necessary for absolute paths).
  - `AiNodeClone`: Since it creates a new node, it will now receive, like the `AiNode` function, the node name and (optionally) the parent node.
- Removed `AtProcInitBounds/procedural_init_bounds` method from the procedural API, since procedurals no longer have user provided bounds (they are always open during initialization). (#5612)

## Arrays

- `AtArray` is now opaque: Any operation involving `AtArray` objects has to be done through its new API, and not accessing its struct members which are now hidden. (#4082)
- `AiArrayAllocate()` does not initialize memory: When an `AtArray` is created with `AiArrayAllocate()`, its contents are no longer initialized to zero and are instead left uninitialized. This results in slightly faster code. (#4507)
- `AiArrayResize()` was added, to resize an `AtArray` in place, preserving its existing data. (#2073)
- Removed `AiArrayModify()`. (#5279)

## Colors

- Removed `AiRGBCreate()` and `AiColor()`, replacing them with actual C++ constructors. For instance, `AtRGBA c = AiRGBACreate(1,0,1,0);` can now be written as `AtRGBA c(1,0,1,0);`. (#2158, #5271)
- Removed `AtColor`: `AtColor` has been deprecated in favor of `AtRGB`. It is now in `ai_deprecated.h`. (#5352)
- Removed `AiColorLerp()` and `AiRGBALerp()` since the preexisting `AiLerp()` already did the same thing. (#5141)
- Removed `AiColorReset()` and `AiRGBAReset()` since it's simpler to just assign `AI_RGB_BLACK` and `AI_RGBA_ZERO`. (#5279)
- Removed `AiColorIsZero`, replace with the equivalent `AiColorIsSmall`. To see if a color is exactly zero use `== AI_RGB_BLACK`. (#5305)
- Removed `AiColorEqual()` in favor of the cleaner `operator==` and `!=` methods. (#5279)
- Removed `AiColorGamma()` and `AiRGBAGamma()`. (#5267)
- Renamed `AI_RGBA_BLACK` to `AI_RGBA_ZERO`, to clarify that its alpha component is zero. (#3928)

## Math

- Removed `AI_TYPE_POINT`: we no longer differentiate between points and vectors and so have removed `AI_TYPE_POINT` and all functions that operated on points when a similar vector function was available. For instance, instead of using `AiNodeSetPnt()`, you can directly use `AiNodeSetVec()`. Search and replace of "Pnt" with "Vec" and "POINT" with "VECTOR" will go most of the way towards updating the code. `AtPoint` and `AtPoint2` have been deprecated and moved into `ai_deprecated.h`. (#5059)
- Removed `AiM4Berp()`, `AiColorBiLerp()`, `AiRGBAbiLerp()`, `AiColorHerp()`, `AiRGBAHerp()`, `AiColorBiHerp()`, `AiRGBAbiHerp()` and `BIHERP()`. (#4646)
- Removed `AiVector()` and replaced them with actual C++ constructors. (#2158, #5271)
- Renamed `MIN()`, `MIN3()`, `MIN4()`, and corresponding `MAX` to `AiMin()` and `AiMax()`, which are overloaded and support 2, 3, or 4 arguments. (#5232)
- Renamed `SQR` to `AiSqr`, `CLAMP` to `AiClamp`, `LERP` to `AiLerp` and `ACOS` to `AiSafeAcos`. (#5595)
- `AtBBox`: The `AtBBox` related functions are now class member functions, so instead of writing something like: `AiBBoxIsEmpty(bb ox)`, we do: `bbox.isEmpty()` (#5446)
- `AtMatrix`: in the Python API, matrix elements are now accessed using the more natural `matrix[i][j]` syntax. (#5242)
- `Nan/inf` checking: We used to have three different names for `Nan/inf` checking of floats, colors and 3D vectors: `IsFinite`, `Corrupted`, and `Exists`. We have now consolidated all of those functions with a consistent name and meaning, following the float version. Therefore, `AiV3Exists()` has been renamed to `AiV3IsFinite()`, and `AiRGB/ACorrupted()` have been renamed to `AiRGB/AIsFinite()`, which reversed their meaning. (#2145)
- `AiBuildLocalFrameShirley()` has been replaced by `AiV3BuildLocalFrame()`, which is faster and has fewer discontinuities.

(#3213)

- AtSampler: AiSamplerSeeded() has been renamed to AiSampler(), so that we now always require a random seed. The sampler now also supports 1D, 2D and 3D sample patterns, instead of only 2D. (#5451)
- AtVector and AtRGB/A comparison operators: AtVector, AtVector2, AtRGB, and AtRGBA now have support for the <, <=, >, and >= operators. After doing a comparison, calling AiAny() on the result will return a bool if any of the element comparisons were true and AiAll() will return true if all the element comparisons were true. For instance, AiAll(AtVector(1,2,3) < AtVector(0,10,20)) will return false since the first element comparison 1 < 0 is false. AiAny(AtVector(1,2,3) < AtVector(1,10,20)) on the other hand returns true. (#2157)
- AiFastPow() added: this fast power function is suitable for cases where speed is more important than accuracy. (#5573)
- Removed AiV3RayPoint(). You now need to do the math yourself, so AiV3RayPoint(out, origin, dir, t) becomes out = origin + dir \* t. (#5089)

## BSDFs

- The BSDF API has been completely revamped to support more advanced and optimized BSDFs, and to be usable for more advanced rendering algorithms. More detailed information is available in the [BSDF documentation](#). (#4783)
- AiBSDFIntegrate() can integrate both direct and indirect light. If only one or the other is needed, passing NULL for the unneeded component will skip it. If both are needed it is fastest to compute them in a single function call. (#5423)
- AiOrenNayarBSDF(), AiMicrofacetBSDF(), AiMicrofacetRefractionBSDF() and AiMetalBSDF() are the built-in BSDFs. AiOrenNayarIntegrate(), AiMicrofacetBTDFIntegrate(), AiDirectDiffuse() and AiIndirectDiffuse() have been removed. (#4783, #5631)
- AiDielectricFresnel() and AiConductorFresnel() have been added. (#5631)
- Renamed AiFresnelWeight() to AiSchlickFresnel(). (#5631)
- Removed Cook-Torrance, stretched Phong, Ashikhmin-Shirley and Ward BRDFs: use Microfacet BRDF instead. (#5161)
- Removed AiHairDirectDiffuseCache() and hair shader diffuse\_cache parameter, use diffuse lighting without cache. (#5161)

## SSS

- Removed AiBSSRDFCubic() and AiBSSRDFGaussian(). AiBSSRDFEmpirical() may be used instead. The cubic radius multiplied by 0.13 approximately matches the mean free path for the empirical BSSRDF. (#5161)
- Removed AiSSSTraceSingleScatter(). The empirical BSSRDF that includes an approximation for single scattering may be used instead, or for brute force volume scattering volume closures may be passed to the interior parameter of AiClosureTransparent() OR AiMicrofacetRefractionBSDF(). (#5161)
- Removed AiSSSEvaluateIrradiance(), use raytraced SSS instead. (#4796)

## Volumes

- Removed AiShaderGlobalsSetVolume\*. Instead volume closures must be used instead, which have the advantage of making volume shaders mixable. (#5503).
- Removed sg->Vo and sg->Ci. Instead AiClosureVolumeAtmosphere() must be used instead. (#4793)
- Renamed volume\_scattering node to atmosphere\_volume to clarify its purpose. (#5495)
- Removed volume shape node with dso parameter. Plugins will instead register custom node types now, for example volume\_open\_vdb. (#5154)
- Volume plugins all now have a bounds\_slack parameter. The plugins are responsible to read that parameter and apply it to the bounds they report in the AtVolumeData struct. (#5831)

## Lights

- AiBSDFIntegrate() can be used to integrate both direct and indirect light. If more low level access is needed, it is still possible to manually implement light loops and multiple importance sampling using the light sampling API. See the documentation for an [example of custom direct light integration](#). However it is strongly suggested to use closures and leave the light sampling to Arnold instead, to benefit from sampling optimizations and support light path expressions. (#5423)
- AiLightsGetSample() now returns an AtLightSample with all light sample data, rather than writing it to shader globals. For light filters the AtLightSample in sg->light\_filter can be used to read and modify the current light sample. (#5423)
- AiLightsTrace() and AiLightsTraceRayTypes() have been added for low level direct light integration. (#5423)
- Removed AiEvaluateLightSample(). (#5423)
- Removed AiLightGetAffectDiffuse() and AiLightGetAffectSpecular(). AiLightGetDiffuse() and AiLightGetspecular() now automatically return zero for when light.affect\_diffuse or light.affect\_specular are off. A new AiLightGetInfluence() utility function can also be used instead of these functions, returning the right influence depending on the shading context and ray type. (#5423)

- `AiLightsIntegrateShadowMatte()` was added to compute more useful shadow mattes. The new matte has color information if needed, correctly weights light occlusion when multiple lights are present and takes into account the surface's BSDF. (#5496)
- Removed `AiLightsGetShadowMatte()`, use `AiLightsIntegrateShadowMatte()` instead. (#5496)

## Plugins

- Procedural and volume plugins now work more similar to other plugins such as shaders, drivers and cameras. They are automatically loaded from paths passed to `AiLoadPlugins()`, `ARNOLD_PLUGIN_PATH` and `options.plugin_searchpath`, registering new node types and defining parameters when they are loaded. The available methods are the same, but the syntax has changed, for details see this [procedural API example](#). (#5154)
- Derived types and type-names of the newly introduced node-like implicit, procedural and volume shape nodes are now exposed through corresponding `AtNodeEntry` functions in the API (#5686)
- Optional `node_plugin_initialize` and `node_plugin_cleanup` methods have been added. These methods can be used for initializing a plugin, if there is some initialization or data to be shared between nodes of the same type. These methods are only called if a node of this type is created. `AiNodeGetPluginData()` can be used to retrieve the plugin data created in `node_plugin_initialize`. (#5154)
- `node_initialize` and `node_update` no longer provide a `params` parameter, and `AiNodeGetParams()` has been removed too. `AiNodeGet*/Set*` and `AiShaderEvalParam*` must now be used to access parameters. For best performance these parameters should be read and stored with `AiNodeGetLocalData()` in `node_update`, and then retrieved though `AiNodeGetLocalData()` in performance critical methods such as shader evaluation and pixel filtering. (#3796, #4853, #5145)
- Removed deprecated parameter creation functions with uppercase types `AiParamterENUM()` etc; use the lowercase versions instead, e.g. `AiParamterEnum()`. (#5182)
- `AiMetadataSet???`: these API functions now operate on a node entry, for symmetry with the `AiMetaDataSet???` functions. The prototype for `node_parameters` has been modified to receive a `AtNodeEntry* nentry` parameter. (#5292)
- `AtParamValue` values are now accessed as a function instead of directly, so for instance, `param.RGB = my_color` is now `param.RGB() = my_color`. (#5194)
- Removed `AiCameraGetLocalData()`, `AiFilterGetLocalData()` and `AiDriverGetLocalData()`. Instead, `AiNodeSetLocalData()` and `AiNodeGetLocalData()` can be used now to attach custom data to camera, filter and driver nodes, the same as for shaders. (#5281)
- Removed `AiCameraDestroy`, `AiFilterDestroy` and `AiDriverDestroy`. They no longer need to be called for custom cameras, filters or drivers, as this de-initialization now happens automatically inside Arnold. (#5278)
- Removed `AtVolumePluginVtable.SampleDeprecated()`, use the `Sample()` method instead. (#4180)
- Removed `AiLoadPlugin()` since `AiLoadPlugins()` does the exact same thing. (#4646)

## Shader Globals

- Added `AiShaderGlobalsGetUniformID()`, for getting a unique per face, subdivision patch, curve or point ID. This uniform ID may for example be used for randomizing colors per hair curve. DCC app plugins can benefit from this by removing the "curve\_id" user data export that is used by certain shaders, and instead using this function, which would reduce memory usage a bit. (#2436)
- Removed `AtShaderGlobals.out_opacity`. Use `AiClosureTransparent()` instead to create mixable shaders with transparency. (#4793)
- Removed `AtShaderGlobals.area`, use the `AiShaderGlobalsArea()` method instead. (#4180)
- Removed image space coordinates from `AtShaderGlobals`: the `sx` and `sy` fields in `AtShaderGlobals` have been replaced with subpixel camera sample coordinates `px` and `py`. If `sx` and `sy` are needed they can be computed using the following code. (#5278)

```
const float sx = -1 + (sg->x + sg->px) * (2.0f / AiNodeGetInt(AiUniverseGetOptions(), "xres"));
const float sy = 1 - (sg->y + sg->py) * (2.0f / AiNodeGetInt(AiUniverseGetOptions(), "yres"));
```

- `AtShaderGlobals.bu` for curve objects has changed to cover the entire width of the curve from 0 to 1. If UVs are specified through `curves.uvs`, they will affect the `u` and `v` texture coordinates. The parametric coordinates `bu` and `bv` always range from 0 to 1 along the width and length of the curve respectively. (#5212)
- `AiShaderGlobalsGetVertexUVs()`: `uvset` parameter added, an empty `AtString()` returns the default UVs. (#5653)
- `AiShaderGlobalsStochasticOpacity()` now applies `geo_opacity` for min pixel width, and should no longer be manually applied by shaders. Instead, calling `AiShaderGlobalsStochasticOpacity()` in the shader will handle the opacity correction required by `min_pixel_width`, and any manual application must be removed to avoid double opacity correction. (#5399)
- For any `opacity` or equivalent parameters in shaders, `AiShaderEvalParamOpacity()` should be used to evaluate it. This enables optimizations for rendering of transparent surfaces. Failing to call this function will result in very slow rendering of opacity-mapped cutouts such as tree leaves or other overlapping transparent surfaces. (#5357)

```
const AtRGB opacity = AiShaderGlobalsStochasticOpacity(sg,  
AiShaderEvalParamOpacity(p_opacity));
```

## Bug fixes

### 5.0.0.0

- #3851 unnormalized lights with zero radius should NOT be treated as normalized
- #3962 Curved motion blur can result in large excursions
- #4108 number of pixels stat is too high
- #5166 fast\_opacity results in noise for low auto\_transparency\_depth
- #5169 change default bump2d height to 1.0 so it matches bump3d
- #5181 kick -info should print default values for matrix parameters too
- #5189 Fix assorted memory leaks
- #5212 Curve barycentric and uv coordinates inconsistency
- #5289 AiNodeClone increases instance overhead memory stats
- #5297 Include detailed reason for failure in append (checkpointing) errors
- #5326 Kick should support -o for raw drivers
- #5367 Subdivs: bad limit surface near vertices connected to both boundaries and creases
- #5368 Subdivs: creased vertices are not correctly handled for UVs or indexed user data
- #5393 Assert triggered when overriding multiple shader assignment with single shader
- #5396 Drivers with append ON should not touch completed files
- #5402 Deep driver: use smaller minimum alpha value for samples
- #5420 Remove incorrect texel offset from persp\_camera.uv\_remap
- #5425 Subdiv: adaptive mode should work with orthographic cameras
- #5427 Subdiv: adaptive mode not working with very small subdiv\_adaptive\_error
- #5430 Corrupted output for high resolution EXR with many AOVs in scanline mode
- #5432 Displacing meshes with high valence vertices (> 256) can crash
- #5448 crash in AtNode::getUserDataMemoryUsage()
- #5454 Ailrradiance returns too much indirect light
- #5457 Mesh light not evaluating UDIMs properly
- #5458 Fix minor memory leaks
- #5459 Account correctly for mesh lights importance table shader calls
- #5468 Different binary outputs from same scene
- #5478 Slow render with infinite velocity values in OpenVDB grids
- #5538 Crashes when accessing the camera in free render mode
- #5551 Portals not working with unconnected skydome light color
- #5562 Skydome light outside portals should be blocked

- #5577 Race condition in procedural population
- #5593 crash in AiTextureAccess when texture\_autotile is changed
- #5643 Array sanity check happens with partially filled array data
- #5644 Crash in procedural creating an object with more than 2 motion keys (5.0)
- #5664 Use GGX instead of Beckman for plastic/metal mode in the utility shader
- #5703 Missing nodes should report an error and abort the render
- #5710 Curve minimum pixel width not working for some camera positions
- #5721 face\_visibility crash
- #5730 Crash using Arnold 5 in C4DtoA from thread other than main
- #5733 Watermarked deep file hangs render
- #5768 AiMakeTx should close file handles of input textures
- #5773 Volumes missing IPR update when changing min and max parameters
- #5792 AiMakeTx should output error messages
- #5794 Add d'Eon and Zinke hair BSDFs in API
- #5815 motion parameters should not be inherited
- #5818 Artifacts with refractive levelset implicits
- #5827 Automatic reloading of a procedural with lights doesn't work
- #5836 Blackbody: should match for all rendering color spaces
- #5837 Refractive dispersion: maximize match for all rendering color spaces
- #5851 Add residual terms to `standard\_hair`
- #5863 Free mode: crash when logging is on
- #5867 UINT AOVs not correctly written out
- #5871 Only warn once for deprecated nodes
- #5881 Allow changing thread\_priority in Windows
- #5896 AiTraceProbe not evaluating opacity
- #4955 OpenEXR driver: only warn about long metadata names if they are actually written out to output file
- #5858 IPR moving in kick doesn't work in flat scenes
- #3928 Rename AI\_RGBA\_BLACK to AI\_RGBA\_ZERO

## 4.2.16.0

### Milestone 4.2.16.0

#### Enhancements

- **Improved oriented curves mode:** The oriented curves mode will now produce an actual 3D surface when rendering, making it more suitable for shapes like grass and feathers. This replaces the previous behavior that was based on the 2D ribbon mode. (#5532)
- **Faster minimum pixel width:** `min_pixel_width` for curves and points is now significantly faster in certain situations. (#5249)
- **Faster maketx:** Rather than use the external `maketx` process to generate .tx files, this can now be done within Arnold using the new `AiMakeTx()` call. This can provide an order of magnitude speedup by processing multiple files in parallel. (#5524)
- **Light portal mode:** skydome lights now have a `portal_mode` parameter with options `off`, `interior_only` and `interior_exterior` to respectively turn off portals, block any light outside portals for interior only scenes, and let light outside portals through for mixed interior and exterior scenes. Blocking light outside portals more predictably reduces noise for interior only scenes. (#5601)
- **New output image metadata:** `arnold/camera/near_clip` and `arnold/camera/far_clip` have been added to output image EXR metadata. (#3175)
- **File handle limits:** We now report the file handle limit used by arnold at the top of the log and if the limit is hit, we print out a list of all the open files in order to aid in determining why there are so many files open at the same time. (#5236, #5237, #5238)
- **Auto-detect threads respects Linux CPU taskset:** On Linux, if the Arnold process is run under a restricted CPU set, such as from `taskset` or `numactl`, then setting the number of threads to 0 will cause Arnold to use the number of cores it's been assigned, instead of the total number of logical cores in the system. (#5283)
- **Upgraded OIIO to 1.7.7:** Arnold now uses OIIO 1.7.7. (#5342, #5000)

#### Incompatible changes

- **Crash reports sent to `stdout`:** Detailed crash reports are now sent to `stdout` instead of `stderr` to prevent lost back trace information. (#5387)

#### Bug fixes

## 4.2.16.0

- #4580 Crash using invalid shader type in certain contexts
- #5462 Deep output should work with AA < 0
- #5489 Closest filter does not work with mattes
- #5499 Bad Z AOV with old volumetric scattering shader and custom filter
- #5568 SIGINT should be passed to parent process
- #5578 Crash with physical sky and non-normalized XYZ vectors
- #5579 Light group RGBA AOVs have incomplete alpha
- #5580 Crash saving .ass file with whole array links
- #5587 Disk light with zero radius rendering wrong
- #5597 Race condition with invalid textures can cause crashes

## 4.2.16.1

- #5543 Skydome light with black texture cannot have color updated
- #5577 Race condition in procedural population
- #5602 Crash when mixing deferred and non-deferred procedurals containing lights
- #5603 Crash when mixing procedural cache with explicit instances
- #5606 AiMakeTxAbort might not clear list of jobs
- #5607 Add python bindings for AiMakeTx
- #5610 Disabled light portals affect the skydome
- #5625 Thread pinning and priority can no longer be set
- #5628 Crash in procedural creating an object with more than 2 motion keys
- #5635 Hang exiting plugins with OpenEXR driver
- #5636 AiMakeTx not handling filenames with spaces

#### 4.2.16.2

- #**2097** Artifacts on implicit surfaces
- #**5472** Fix uniform implicit solver for non-levelset fields
- #**5622** N AOV does not include bump information when shadow\_matte is computed too
- #**5662** Warn if missing SSE4.1 support, instead of aborting
- #**5663** AiTextureLoad not using texture\_searchpath
- #**5665** Downgrade to SSE4.1
- #**5697** missing UDIM textures do not scale
- #**5704** Same UDIM texture accessed through different texture nodes can be slow
- #**5705** UDIM textures with ignore\_textures set can be very slow

#### 4.2.16.3

- #**5645** Fix temporary hangs under Windows
- #**5720** Crash when removing unloaded deferred procedurals
- #**5721** face\_visibility crash
- #**5723** Crash when modifying matrix while reloading a procedural
- #**5733** Watermarked deep file hangs render
- #**5743** Write time message for .ass files too short
- #**5759** Always scale past 64 threads in Windows
- #**5768** AiMakeTx should close file handles of input textures
- #**5773** Volumes missing IPR update when changing min and max parameters

## 4.2.16.1

### Milestone 4.2.16.1

#### Bug fixes

- #5543 Skydome light with black texture cannot have color updated
- #5577 Race condition in procedural population
- #5602 Crash when mixing deferred and non-deferred procedurals containing lights
- #5603 Crash when mixing procedural cache with explicit instances
- #5606 AiMakeTxAbort might not clear list of jobs
- #5607 Add python bindings for AiMakeTx
- #5610 Disabled light portals affect the skydome
- #5625 Thread pinning and priority can no longer be set
- #5628 Crash in procedural creating an object with more than 2 motion keys
- #5635 Hang exiting plugins with OpenEXR driver
- #5636 AiMakeTx not handling filenames with spaces

## 4.2.16.2

### Milestone 4.2.16.2

#### Bug fixes

- #2097 Artifacts on implicit surfaces
- #5472 Fix uniform implicit solver for non-levelset fields
- #5622 N AOV does not include bump information when shadow\_matte is computed too
- #5662 Warn if missing SSE4.1 support, instead of aborting
- #5663 AiTextureLoad not using texture\_searchpath
- #5665 Downgrade to SSE4.1
- #5697 missing UDIM textures do not scale
- #5704 Same UDIM texture accessed through different texture nodes can be slow
- #5705 UDIM textures with ignore\_textures option set can be very slow

## 4.2.16.3

### Milestone 4.2.16.3

#### Bug fixes

- #5645 Fix temporary hangs under Windows
- #5720 Crash when removing unloaded deferred procedurals
- #5721 face\_visibility crash
- #5723 Crash when modifying matrix while reloading a procedural
- #5733 Watermarked deep file hangs render
- #5743 Write time message for .ass files too short
- #5759 Always scale past 64 threads in Windows
- #5768 AiMakeTx should close file handles of input textures
- #5773 Volumes missing IPR update when changing min and max parameters

## 4.2.16.4

### Milestone 4.2.16.4

#### Bug fixes

- #5792 AiMakeTx should output error messages
- #5827 Automatic reloading of a procedural with lights doesn't work
- #5881 Allow changing thread\_priority in Windows
- #5925 Properly handle cases where cpuset is smaller than the detected number of cores
- #5926 ignore\_motion\_blur with non-zero reference\_time does not work with a polymesh with normals
- #5953 Wrong stats after destroying shape nodes
- #5858 IPR moving in kick doesn't work in flat scenes

## 4.2.15.0

### Milestone 4.2.15.0

#### Enhancements

- **Implicit surface plugins:** Through the volume plugin API, volume plugins can now provide implicit surface fields. The implicit node has been extended to use these plugins with a new dso parameter, as well as a field\_channel parameter to specify which channel from the volume plugin contains the field information. Usually these plugins will provide a signed distance field or level set, so be sure to set the solver to "levelset" in that case on the implicit node. (#5365)
- **Light portals:** Skydome lights can now use portals to reduce noise for interior scenes, where light comes into through relatively small openings. Light portals are **quad\_light** nodes with parameter **portal on**, and instead of emitting light these will be used to guide skydome light sampling. Light portals must be placed to cover all windows, doors and other openings through which skydome light comes into the scene. (#4720)
- **diffuse\_albedo AOV:** Core shaders will set the built-in diffuse albedo AOV. Custom shaders can be modified to set it if needed. (#5485)
- **shadow\_matte AOV:** Makes available the value from the API AiLightsGetShadowMatte call in a built-in AOV. (#5486)

#### API additions

- **Volume plugin gradients:** Volume plugins can now set an optional callback to calculate the gradient of scalar (float) channels. This is particularly used by the new implicit node for implicit surfaces to find the surface normal. It is also sufficient to simply return AI\_V3\_ZERO if the gradient is unknown, and the gradient will be calculated by Arnold instead, although possibly with less efficiency. (#5365)
- **String length limits:** There are no longer any limits on parameter name, metadata name, message and filename lengths. (#3320)

#### Incompatible changes

- **Standard shader:** Opacity is now clamped to 0 below 1e-6 instead of 1e-4 (#5374)

#### Bug fixes

##### 4.2.15.0

#4900 Long file paths silently fail or cause crashes

#5350 Rendering camera should be included in random number seeds

#5374 Very low opacity clamped to 0

#5484 Always output actual shading normal in 'N' AOV

#5507 spot light bounds doesn't factor in lens radius

#5519 sg.Rd and ray.dir are sometimes not properly normalized

## 4.2.15.1

### Milestone 4.2.15.1

#### Bug fixes

- #5537 Fix built-in `shadow\_matte` AOV to match the common library shader `shadow\_matte`
- #5551 Portals not working with unconnected skydome light color
- #5554 AiNodeDestroy sometimes returns wrong value when destroying procedural nodes
- #5561 Destroying a procedural node doesn't always release the node name
- #5562 Skydome light outside portals should be blocked

## 4.2.14.0

### Milestone 4.2.14.0

#### Enhancements

- **Improved SSS:** Subsurface scattering has been modified to leak less light into areas where it shouldn't and properly contribute to indirect light. This improves results around the nose or mouth in typical head models for example. Previously SSS was ignored for secondary GI bounces or contributing to more bounces than specified with GI\_diffuse\_depth. (#4174, #2978, #4648)
- **Better defaults in standard shader:** The empirical SSS profile and GGX microfacet specular distribution are now used by default. (#5317)
- **Automatic reloading of procedurals:** When a procedural geometry node's dso or data parameters are modified during an interactive session, the procedural will now be automatically reloaded, and the old contents cleared, avoiding the need for manually destroying and creating a new procedural. (#5190)
- **Faster rendering of few buckets:** Rendering a small image will now scale to all available CPU cores regardless of bucket size. (#4647)
- **Maximum number of threads:** The maximum number of threads has been increased from 128 to 256. (#5349)
- **Updated OIIO to 1.5.24:** This new version comes with some optimizations and bug fixes. (#5139)

#### API additions

- **AiSetLicenseString():** Added a new API function for setting up a license string enclosed within angle brackets, which is the LICENSE line in a node-locked license file. Client code could now use this function and forget about setting up any environment variables, such as solidangle\_LICENSE. (#5259)  

```
AiSetLicenseString("<LICENSE solidangle arnold YYYYMMDD 10-Jun-2016 uncounted hostid=... _ck=... sig=...>");
```
- **Python bindings for texture API:** The Python API now exposes the following functions in ai\_texture.py (#5293):
  - AiTextureGetResolution()
  - AiTextureGetNumChannels()
  - AiTextureGetChannelName()
  - AiTextureGetFormat()
  - AiTextureGetBitDepth()
  - AiTextureGetMatrices()
  - AiTextureInvalidate()

#### Incompatible changes

- **Disabled RLM broadcasting:** We have disabled RLM broadcasting, which removes the two-second connection timeout delay due to the RLM client attempting to broadcast to find a license server in the LAN. From now on, the RLM license server must always be specified using the solidangle\_LICENSE environment variable, even if it's localhost. (#4535)
- **Removed AiLicense{Set|Get}Attempts() and AiLicense{Set|Get}AttemptDelay():** We have removed the "retry" RLM license request ability. (#2893)

#### Bug fixes

Ticket	Summary
#5327	Windows signal handlers not cooperating with parent process
#4647	Inefficient use of CPU threads when using a low number of buckets
#4648	SSS not respecting diffuse GI depth correctly
#5084	Fix sampling regression introduced in recent sampling refactoring
#5157	Multilayer EXR should properly output INT AOVs when other types are present
#5165	Hang when repeatedly adding and removing nodes
#5167	Crash when destroying a scene with cloned quad lights
#5170	Crash when destroying a procedural used by the procedural cache
#5173	Crash with volumes when light volume_samples is zero
#5179	Deep driver: fix uninitialized variables
#5184	Destroying any cached procedural will invalidate all copies
#5191	EXR driver: autocrop corrupts the image when HALF AOVs are present
#5192	Deep driver: uniform treatment of layer attributes
#5193	Hang when destroying and re-creating procedurals with cache enabled
#5201	Procedurals containing only shaders are not destroyed correctly
#5216	Empirical SSS profile rendering incorrectly with low radius
#5217	Matte object AOV filtering wrong with deep EXR
#5227	poorly scaling codepath in catclark subdivision
#5228	SSE4.2 not always correctly detected
#5233	allHair transmission component incorrect
#5234	Expand environment variables for volume.dso
#5239	Metadata files don't allow 'RGB' as attribute or metadata name
#5240	Deep driver: Surface samples should merge when inside a volume
#5248	Invalid scene bounds with min_pixel_width in free mode
#5256	Proc inheritance not respected for bump shaders
#5260	Spot light gobo texture blurred too much in volumes
#5274	Mesh lights within procedurals are not correctly transformed
#5280	Windows-only texture lookup crash for constant-valued .tx images
#5301	AiTextureAccess crash with multiple Arnold sessions
#5303	Crash when saving TIFF if resolution is not a multiple of tile size
#5304	refraction_opacity should track through multiple refractions
#5308	Wrong varying and indexed interpolation for integer user data types
#5319	Stray info messages at lower log verbosity
#5320	Rare failed assert in LightRadiance() with atmospheric scattering
#5329	Deep AOV sample iterator losing mixed atmosphere and surface samples
#5332	Atmosphere RGBA AOV missing from deep EXR files
#5337	server log entries because maketx tries to access nonexistent RPATH
#5341	Override nodes applied to instances not working correctly
#5302	Fixed some wrong geometry stats after destroying scene nodes
#5358	Do not output warning for zero scaled transforms

## 4.2.14.1

### Milestone 4.2.14.1

#### Bug fixes

Ticket	Summary
#5329	Deep AOV sample iterator losing mixed atmosphere and surface samples
#5367	Subdivs: bad limit surface near vertices connected to both boundaries and creases
#5368	Subdivs: creased vertices are not correctly handled for UVs or indexed user data
#5377	Fix memory leak and overhead due to samplers
#5383	delay in opening kick window
#5391	Render checkpointing incorrectly computes channel offsets
#5396	Drivers with append ON should not touch completed files
#5402	Deep driver: use smaller minimum alpha value for samples
#5403	lights and ginstances incorrectly only use the first matrix transform key when curved_motionblur is disabled

## 4.2.14.2

### Milestone 4.2.14.2

#### Bug fixes

Ticket	Summary
#5414	Rare crash in Linux when Arnold is dynamically loaded
#5417	node update is very slow when there are many nodes
#5420	Remove incorrect texel offset from persp_camera.uv_remap
#5425	Subdiv: adaptive mode should work with orthographic cameras
#5427	Subdiv: adaptive mode not working with very small subdiv_adaptive_error
#5430	Corrupted output for high resolution EXR with many AOVs in scanline mode

## 4.2.14.3

### Milestone 4.2.14.3

#### Bug fixes

Ticket	Summary
#5189	Fix assorted memory leaks
#5418	parallel node init/update performance regression
#5432	Displacing meshes with high valence vertices (> 256) can crash
#5450	Crash when setting parameters on a disabled procedural
#5457	Mesh light not evaluating UDIMs properly
#5458	Fix minor memory leaks
#5433	minor bug fixes

## 4.2.14.4

### Milestone 4.2.14.4

#### Bug fixes

Ticket	Summary
#5462	Deep output should work with AA < 0
#5468	Different binary outputs from same scene
#5479	Crash when calling AiNodeDestroy on a procedural with invalid dso path
#5480	Procedural failing when re-setting the same dso
#5482	artifact near center of spherical light with volume scattering
#5507	spot light bounds doesn't factor in lens radius
#5509	Mesh light incorrect render or crash with deformation motion blur
#5510	Procedural children not correctly removed when destroying procedural
#5474	stack overflow at high sampling rates

## 4.2.13.0

### Milestone 4.2.13.0

#### Enhancements

- **SSE4.2 optimizations:** We have raised our minimum CPU requirements from CPUs needing to support the SSSE3 instruction set to SSE4.2. This allows us to start taking advantage of the additional SSE4 instructions to further optimize Arnold. Already we are seeing a slight speedup in several areas. For instance, the Hosek Sky shader is now 5% faster. (#2625, #5051, #5053, #5056)
- **Faster OpenVDB motion blur:** Rendering of motion blur on CPUs with many threads is now faster. The speedups depend on the specific scene and the CPU, but we have often seen 2x faster, and even 10x faster in some cases. Additionally, rendering of OpenVDB files with separate velocity grids for the XYZ components is now also up to 2x faster, so that merging the grids into a single vector velocity grid is no longer required for good performance. Merging the grids in advance still helps to make the file loading a little faster. (#5103)
- **Faster gobo shader:** The number of shader evaluations in the gobo light filter has been greatly reduced. We are seeing 20% faster renders in simple scenes with a light and a gobo. (#5070)
- **Faster textures:** AiTextureAccess() is now much closer (though still not equal) in performance to AiTextureHandleAccess(). We have also made AiTextureHandleAccess() faster in certain situations. (#3192, #3699, #5069, #5068)
- **Default texture cache size increased to 2GB:** The default setting for options.texture\_max\_memory\_MB has been increased from 1GB to 2GB. This can help improve performance on texture heavy scenes. (#5101)
- **Quad and disk light spread:** Quad and disk lights can now emit light focused in the direction along the normal. The default spread value of 1 gives diffuse emission, while lower values focus the light more, until it becomes a laser like beam at value 0. Low spread values can be noisier than the default high spread, so be careful when using them. (#5119)
- **Photometric light radius:** Photometric lights now include a radius parameter with which softer shadows may be obtained in a manner similar to point lights or spot lights. Defaults to 0, which matches this light's original behavior. Note that IES files only store far-field photometric values, so it's meaningless to use them with area lights, and abusing the new radius parameter can make things look strange. The intent of adding this radius is to achieve slightly soft shadows rather than razor sharp shadows in a manner that is acceptable for VFX use. (#5121)
- **More shader parameter conversions allowed:** It is now possible to link a shader that outputs a float to a shader that takes in an integer type and vice versa. Likewise, several other useful types of conversions are also now allowed. (#5058, #5095)
- **Deep EXR driver support for append / checkpointing:** driver\_deepexpr now supports append on to skip rendering buckets that were saved in a previous interrupted render. Currently this only works for tiled mode, not scanline. Note that since Nuke only works with scanline deep files, if your output deep files are tiled you will have to convert them with oiotool input\_tiled\_deep.exr --scanline output\_scanline\_deep.exr (#4863)
- **Improved deep EXR driver support for volumes:** Deep volumes are now written with higher precision using less samples, which results in much more accurate and smaller deep EXR files, with size savings of 50 to 75%. (#5102, #5113, #5114, #5120, #5135)
- **Fewer warnings:** A number of pedantic warnings have been removed in order to declutter log files: nonexistent shader paths, camera matrix overriding handedness, deprecated RGB AOV, empty polymesh node, UV array compression, ignoring mesh with all faces invisible, user normals on subdivs, too many messages of given type. (#5042)
- **Watermarks for shaderballs:** Shaderball previews in our plugins (e.g. MtoA) no longer show the "arnold" watermark. (#5074)
- **Updated to RLM 12.0BL2:** We have upgraded the license server and the external library controlling the licensing subsystem from version 11.3BL1 to 12.0BL2, a more stable release fixing various crashes, bugs, hangs and memory leaks. (#5014)

#### API additions

- **AiNodeGetParent():** Return the AtNode\* procedural (i.e. the parent node) that created the input node. (#5004)
- **AiParamTypeConvertible():** Added API function to determine if a parameter src\_type can be converted into the dst\_type parameter. (#5097)
- **AiMsgGetLogFileFlags():** Added AiMsgGetConsoleFlags() and AiMsgGetLogFileFlags() API functions to get the currently set logging flags. These now match the corresponding AiMsgSetLogFileFlags() and AiMsgSetLogConsoleFlags() API functions (#5109)
- **UVs in quad light filters:** Light filter shaders on quad lights can now use sg->u and sg->v to retrieve the UV coordinates on the surface of the quad light. (#5122)

#### Incompatible changes

- **Require SSE4.2 CPU:** We now require CPUs that support (at least) SSE4.2 instructions. On older computers that do not support these instructions, Arnold will immediately abort on startup. (#2625, #5053)
- **Internal reflections in standard shader:** The enable\_internal\_reflections parameter previously did not ignore all internal reflections, and incorrectly ignored some other reflections, which is now fixed. (#5136)
- **Specular roughness:** When using very low or zero roughness in the standard shader or microfacet API functions, reflections will be sharper than before, where previously they were too blurry. (#5067)

#### Bug fixes

Ticket	Summary
#2654	Implement a more robust ray-cone intersector
#3699	Filtered texture result cache slows down evaluation in corner cases
#5027	AOVs of type AI_TYPE_NODE do not work
#5028	shader AOV does not work
#5033	Deep EXR light AOVs crash
#5034	Bump should work regardless of UV quality
#5035	Deep EXR light AOVs not composed correctly
#5039	Problems when first element in the list of disp_map shaders is NULL
#5041	Incorrect deep EXR render with matte surfaces behind volumes
#5046	Light AOV name change to use '_' instead of '.' as separator
#5054	Crash when comparing compressed arrays
#5064	Incorrect subdivision for soft creases on boundaries
#5067	Standard shader with specular roughness zero is not perfectly sharp
#5071	textures with empty filenames should be an error
#5078	Misleading warnings when loading invalid volume and procedural DSOs
#5083	UDIMs with linked UVs or custom uv set are using wrong UDIM column
#5086	Wrong transparency in front of opaque matte surfaces
#5091	NaNs in Ward-Duer and stretched Phong BSDF sampling weights due to division by zero
#5092	cached indirect diffuse values are incorrect
#5096	rectangular quad light check is overly aggressive
#5099	Improve sample merging for deep volumes
#5102	Deep volume output accuracy should not worsen with more AA samples
#5108	Procedurals create duplicate objects
#5129	Gobo light filters connected to a float slidemap result in red colors
#5132	crash when rendering large un-tiled EXR images
#5134	Bad texture derivatives in rare cases from sphere objects
#5136	Standard shader enable_internal_reflections not working for specular
#5057	missing ai_shaderglobals.h include

## 4.2.13.1

### Milestone 4.2.13.1

#### Bug fixes

Ticket	Summary
#5157	Multilayer EXR should properly output INT AOVs when other types are present
#5165	Hang when repeatedly adding and removing nodes
#5167	Crash when destroying a scene with cloned quad lights
#5170	Crash when destroying a procedural used by the procedural cache
#5173	Crash with volumes when light volume_samples is zero

## 4.2.13.2

### Milestone 4.2.13.2

#### Bug fixes

Ticket	Summary
#5184	Destroying any cached procedural will invalidate all copies

## 4.2.13.3

### Milestone 4.2.13.3

#### Bug fixes

Ticket	Summary
#5179	Deep driver: fix uninitialized variables
#5191	EXR driver: autocrop corrupts the image when HALF AOVs are present
#5192	Deep driver: uniform treatment of layer attributes
#5193	Hang when destroying and re-creating procedurals with cache enabled

## 4.2.13.4

### Milestone 4.2.13.4

#### Bug fixes

Ticket	Summary
#5201	Procedurals containing only shaders are not destroyed correctly
#5216	Empirical SSS profile rendering incorrectly with low radius
#5217	Matte object AOV filtering wrong with deep EXR
#5227	poorly scaling codepath in catclark subdivision
#5233	aiHair transmission component incorrect
#5234	Expand environment variables for volume.dso
#5240	Surface samples should merge when inside a volume

## 4.2.13.5

### Milestone 4.2.13.5

#### Bug fixes

Ticket	Summary
#4534	2-second timeout when (failing) broadcasting RLM servers
#5228	SSE4.2 not always correctly detected
#5239	Metadata files don't allow 'RGB' as attribute or metadata name

## 4.2.13.6

### Milestone 4.2.13.6

#### Bug fixes

Ticket	Summary
#5248	Invalid scene bounds with min_pixel_width in free mode
#5256	Proc inheritance not respected for bump shaders

## 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 or GI\_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 parameter sss\_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

## 4.2.12.1

### Milestone 4.2.12.1

#### Bug fixes

Ticket	Summary
#5027	AOVs of type AI_TYPE_NODE do not work
#5028	shader AOV does not work
#5033	Deep EXR light AOVs crash
#5035	Deep EXR light AOVs not composed correctly
#5041	Incorrect deep EXR render with matte surfaces behind volumes
#5046	Light AOV name change to use '_' instead of '.' as separator

## 4.2.12.2

### Milestone 4.2.12.2

#### Bug fixes

Ticket	Summary
#5054	Crash when comparing compressed arrays

## 4.2.12.3

### Milestone 4.2.12.3

#### Bug fixes

Ticket	Summary
#5078	Misleading warnings when loading invalid volume and procedural DSOs
#5083	UDIMs with linked UVs or custom uv set are using wrong UDIM column

## 4.2.11.0

### Milestone 4.2.11

#### Enhancements

- **Russian Roulette:** The standard and lambert shaders now use Russian Roulette termination to more efficiently render with high GI depth. For AA samples 5 or higher the increase in noise is typically very small. Indoor scenes with high GI depth will benefit the most, but also scenes with lots of glass and high refraction/reflection depth. In such scenes we have measured between 1.5x and 5x faster renders. (#4901)
- **Global shader override:** It is now possible to override the shader for all objects in the scene by specifying an existing shader in the new global option `shader_override`. (#4909)
- **Search paths:** The procedural and shader search paths can now use both `:` and `;` characters as separators for multiple paths, on all platforms. Texture search paths already supported this. (#4897)
- **Custom attributes in deep EXR:** Just like the regular EXR driver, the deep EXR driver now also supports custom metadata/attributes via the new parameter `driver_deepexpr.custom_attributes`. (#4915)
- **Render options and stats in EXR metadata:** Several global render options, such as sample settings and ray depths, are now stored in the image file as EXR metadata. We also store a few render stats, such as date, used memory, number of polygons and curve segments. These EXR attributes use a path-like metadata layout, e.g. `"arnold/options/AA_samples"`, `"arnold/stats/memory/peak"`, or `"arnold/host/hw"`. We might add a few extra attributes in future releases, and perhaps rename some existing attributes based on customer feedback. (#4849, #4860)
- **Env var expansion in procedural nodes:** The procedural.dso parameter now supports expansion of environment variables delimited by square brackets, similar to the env var expansion in searchpaths in the options node. (#4937)
- **Removed size limit on node and metadata names:** Node names, node entry names, and metadata item names no longer have any size limitations. (#4932)
- **Report memory for smooth derivs:** The memory usage summary for polymeshes now includes a separate line to account for subdiv\_smooth\_derivs storage. (#4925)
- **Upgraded OIIO to 1.5.20:** We have upgraded the OpenImageIO library used for reading texture maps from 1.5.15 to 1.5.20. There have been many changes between these two versions, including many little bug fixes and optimizations. (#4739, #4735, #4855, #4864)

#### API additions

- **AtString version of AiNodeEntryGetName():** Added an alternate version of `AiNodeEntryGetName()` called `AiNodeEntryGetNameAtString()` that returns an `AtString` name instead of a `char*` name. (#4917)

#### Incompatible changes

- **Renamed sss and volume sampling options:** The global options `volume_indirect_samples` and `sss_bssrdf_samples` have been renamed to `GI_volume_samples` and `GI_sss_samples` respectively, for consistency with the other existing sampling options (`GI_diffuse_samples` etc). The old names will still work, as we have added them as deprecated synonyms, but will result in warnings in the log files. We recommend that any client code (such as proprietary DCC plugins) is changed to use the new names. (#4940)
- **Default sss samples:** The default value of the newly renamed `GI_sss_samples` option has been changed from 0 to 2. Although rare, this will change the look of old scenes that contain SSS objects but where the SSS samples setting was left at its default of 0. (#4940)

#### Bug fixes

Ticket	Summary
#4886	render crash on a scene with lots of ginstance
#3196	Wrong scene creation time stats for interactive rendering
#4735	Invalid alpha channel for image textures without alpha in indirect bounces
#4897	Search path splitting inconsistent between parameters and platforms
#4908	texture flush happens a frame too late in linux and OS X
#4910	AiMicrofacetBTDFIntegrate rendering with wrong IOR
#4911	Rare numerical precision artifacts causing noisy bumpmapping
#4914	UDIM tile selection needs to handle bad barycentric coordinates
#4918	Incorrect render with matte surfaces behind volumes
#4921	Deep EXR: preserve float or RGB ID values when tolerance is zero
#4924	Hostname missing in the logs on some Linux distributions
#4942	Nonexisting shader path warnings for drive letters on Windows
#4943	kick warns about gamma/dither when writing to EXR via the '-o' option
#4898	rays/pixel stat not outputting correct value when there were 0 rays/pixel
#4899	AtString version of AiNodeLookUpUserParameter crashes

## 4.2.11.1

### Milestone 4.2.11.1

#### Bug fixes

Ticket	Summary
#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
#4954	Render metadata should have short names for maximum OpenEXR compatibility

## 4.2.11.2

### Milestone 4.2.11.2

#### Bug fixes

Ticket	Summary
#4959	Print better info when getting killed by external process, but not crashing
#4997	Bounding box given by options.curved_motionblur is clipped
#4978	Kick window unexpectedly closes on OS X 10.11

#### 4.2.11.3

##### Milestone 4.2.11.3

###### Bug fixes

Ticket	Summary
#5002	crash if there are more than 100 UDIM rows

## 4.2.10.0

### Milestone 4.2.10

#### Enhancements

- **Reduced OpenVDB memory usage:** In the OpenVDB DSO provided with our DCC plugins, voxel storage is now optimized to reduce memory usage by about 45%. Depending on the scene this may also reduce render time up to 10%. Additionally, slow .vdb file loading from network drives on Windows has been fixed, and we have improved the accounting of OpenVDB memory usage in the stats. (#4858, #4869, #4870, #4871)
- **Autobump on polymeshes:** The autobump feature, enabled by disp\_autobump, is now supported on polymeshes with no subdivision. Previously it required subdivision to work. (#4885)
- **Faster subdivision:** Faster exact (limit) normals and tangents, in certain cases, subdivision\_smooth\_derivs speedups up to 50%, smoothing speedups up to 25%. (#4867, #4868, #4878)
- **Faster AiVCeilNoise3/4:** The vector cell noise API functions AiVCeilNoise3() and AiVCeilNoise4(), which are used in e.g. MayaNoise type "billow", are now about 1.6x faster. (#4889)
- **EXR long names:** Attribute and channel names up to 255 chars are now supported in EXR files. Arnold will warn if long names are used since the output files are not compatible with reader applications that use EXR libraries older than 1.7.0. (#4862)
- **Log improvements:** The system/unaccounted line under the render time log category has been updated to actually report the unaccounted time, which we define as the total render time reported minus all the other render related time stats. Also, the rays/pixel stats printed at every 5% progress message is now more accurate, computed since the previous 5% message, and is no longer a "running average" of the rays/pixel for the entire image. (#4843, #4894)

#### API additions

#### Incompatible changes

- **AiNodeReset and AiNodeResetParameter clear user data:** AiNodeReset will not only reset all built-in parameters to their default values and remove any node links, but will also remove any user data for that node. Additionally, AiNodeResetParameter will remove a single user data element if given its name (#4857)
- **Output driver append mode:** To prevent wasted work, Arnold will now error out when some drivers are in append mode and others are not, or do not support it. Also, when in append mode, display viewers (such as the one in kick, MtoA, etc) will not force a full render and instead will just display the new buckets as they are rendered. (#4841)
- **Non-planar 'Z' AOV:** The Z-depth AOV for custom cameras or non-planar perspective cameras now corresponds to distance along the camera ray. This had been broken since 4.2.4.0, and before that had a different value for custom cameras. For those cameras, 'Z' used to be distance from 'P' to camera center, which could sometimes result in decreasing Z values along the ray. (#4722)
- **Scaled camera transforms:** Only rotation and translation are allowed on camera transforms. Scaling or skew will be ignored, and a warning printed out. This was previously not done consistently, causing unexpected results with DOF. (#2260)
- **Faster AiVCeilNoise3/4:** The "seed" for the random numbers in these functions has effectively changed, so while the type of noise is the same, the specific output given is now different. (#4889)

#### Bug fixes

Ticket	Summary
#2260	Camera scaling should always be ignored
#4722	Incorrect Z values for custom cameras with offset ray origin
#4841	Error out on append driver mismatch
#4846	detect and report errors while reading .ass* files
#4854	Linear basis curves crash with motion blur and varying radius
#4865	Adaptive Subdivs: uninitialized memory while computing exact derivatives
#4866	Blocky artifacts with matte objects
#4868	Ignore subdivision_smooth_derivs if there are no UV coordinates
#4869	OpenVDB file loading slow on Windows network drives
#4871	OpenVDB memory usage reported as unaccounted in stats
#4876	Shadow precision issue with object and mesh light far away from origin
#4888	Bad smoothed normals for linear subdivs or catclark subdiv creases with multiple position keys
#4890	Let bump2d and bump3d work with very small bump_height
#4887	Failed AtArray allocation not returning NULL

## 4.2.10.1

### Milestone 4.2.10.1

#### Bug fixes

Ticket	Summary
#4897	Search path splitting inconsistent between parameters and platforms
#4910	AiMicrofacetBTDFIntegrate rendering with wrong IOR
#4898	rays/pixel stat not outputting correct value when there were 0 rays/pixel
#4899	AtString version of AiNodeLookUpUserParameter crashes

## 4.2.10.2

### Milestone 4.2.10.2

#### Bug fixes

Ticket	Summary
#4911	Rare numerical precision artifacts causing noisy bumpmapping
#4914	UDIM tile selection needs to handle bad barycentric coordinates
#4918	Incorrect render with matte surfaces behind volumes
#4921	Deep EXR: preserve float or RGB ID values when tolerance is zero
#4924	Hostname missing in the logs on some Linux distributions

## 4.2.9.0

### Milestone 4.2.9

#### Enhancements

- **Refractive dispersion:** The standard shader now supports chromatic dispersion through refraction. The new parameter dispersion\_abbe specifies the Abbe number of the material, which describes how much the index of refraction varies across wavelengths. For glass and diamonds this is typically in the range 10 to 70, with lower numbers giving more dispersion. The default value is 0, which turns off dispersion. The chromatic noise can be reduced by either increasing the global AA samples, or the refraction samples. (#4835)
- **Volumetric scattering AOVs:** Direct and indirect scattering in volumes can now be rendered separately using the new volume\_direct and volume\_indirect AOVs. (#4806)
- **Perspective OpenVDB volumes:** The accompanying OpenVDB volume DSO provided with our DCC plugins has now been optimized to render perspective grids, also known as frustum buffers. The speedup depends on the specific perspective transformation, but we have seen 4x-6x faster rendering in tests. (#4827)
- **Faster volumes:** Volume lookups with multiple channels render 5% faster in various tests. (#4516)
- **Faster curves:** Curves in thick mode now render 2-20% faster. Curves with non-zero min\_pixel\_width now render up to 2x faster. (#4823, #4824)
- **Faster deep EXR:** The driver\_deepexpr node has been optimized, resulting in up to 25% faster rendering, most notably when using volumes. (#4811)
- **Faster subdivision:** Several optimizations were added to the Catmull-Clark subdivision engine. Adaptive subdivision mode is now 4% faster, and smooth derivatives (as needed by anisotropic surface shading) are 27% faster while using 8% less memory. (#4832, #4837)
- **Faster opacity:** The arnold core shaders (lambert, standard, etc...) now make use of AiShaderGlobalsApplyOpacity() which can result in significant speedups when shading with partial opacity, or when using opacity-mapped tree leaves. This optimization is controlled by options.enable\_fast\_opacity. We have also improved AiShaderGlobalsApplyOpacity() so that it is faster and higher quality than before. User shaders are advised to make use of this function in order to benefit from the faster opacity using code such as the example below. (#4434, #4775, #4800, #4814)

```
AtColor opacity = AiShaderEvalParamRGB(p_opacity);
if (AiShaderGlobalsApplyOpacity(sg, opacity))
    return;
opacity = sg->out_opacity; // the new opacity is contained in sg->out_opacity
```

- **Driver time stats:** Output driver time is now reported in the render time section of the log file stats. This can be used to detect bottlenecks related to deep image output, excessive number of AOVs or networking issues. (#4840)

#### API additions

- **AtString volume API:** AiVolumeSample functions in shaders and the volume plugin Sample method now accept AtString channel name parameters, for better performance. (#4516)
- **Separate direct and indirect SSS:** Variants of the SSS/BSSRDF shading and lighting API that return both the direct and indirect results separately now exist. These new API functions, denoted by the Separate suffix, accept two variables by reference in which the direct and indirect components are stored separately, instead of returning their sum as the function's result. The following functions were added:

```
AI_API void AiBSSRDFCubicSeparate(const AtShaderGlobals* sg, AtRGB& direct, AtRGB& indirect, const float* radius, const AtColor* weight, unsigned int num = 1);
AI_API void AiBSSRDFGaussianSeparate(const AtShaderGlobals* sg, AtRGB& direct, AtRGB& indirect, const float* variance, const AtColor* weight, unsigned int num = 1);
```

The built-in standard shader and the external skin shader have also been extended to make use of this new API by means of separate direct and indirect SSS AOVs. (#3217)

- **AiTextureLoad() EXPERIMENTAL API:** We have introduced a new API that allows for easily reading an entire texture file into a pre-allocated image buffer. The main use for this is for Maya/MtoA to support .tx texture maps. This can be called outside of AiBegin/AiEnd() blocks so that it can be used at any time. The corresponding AiTextureGetResolution(), AiTextureGetNumChannels(), AiTextureGetChannelName(), AiTextureGetFormat(), AiTextureInvalidate(), and AiTextureGetMatrices() can now also be called outside of AiBegin/AiEnd() blocks. Note that this is currently an EXPERIMENTAL API addition, so we might modify this in the near future. As such, code that uses this might stop working and need to be updated with newer versions of Arnold. (#4831)

```
AI_API bool AiTextureLoad(const AtString filename, const bool use_float, void* image);
```

## Incompatible changes

- **Depth of field:** Perspective cameras have a new `persp_camera.flat_field_focus` attribute which is set to true by default to match the standard thin lens camera model. This prevents overblurring away from the optical axis. Some renders might change, specially with wide FOV angles or very shallow DOF. Set `persp_camera.flat_field_focus` false to get the previous behavior. (#4810)
- **Opacity in utility shader:** utility shaders with opacity will now render less transparent than before, previously it was incorrectly rendering too transparent. The new opacity behaviour better matches other shaders like standard and lambert. (#4836)
- **Bump and object scale:** The bump2d, bump3d shaders and the `AiShaderGlobalsEvaluateBump()` API function now take into account object scale by default. Previously the bump displacement height would not scale along with the object, which was inconsistent with displacement and autobump. The new global option `bump_space` is set to `object` by default, but can be set to `world` for backwards compatibility. This new global option will likely be removed in the next API-breaking release. (#4784)

## Bug fixes

Ticket	Summary
#2807	subdiv_smooth_derivs doesn't work in polymeshes with adaptive subdivision
#4523	Free mode crash using autobump and multiple light loops
#4546	Random procedural transform resets with options.parallel_node_init
#4627	load_at_init procedural inside a delayed load procedural hangs Arnold
#4784	Bump2d is not affected by scale
#4797	Destroy the contents when destroying a procedural node
#4799	Wrong procedural contents reported in the log file when writing to .ass
#4801	Potential crash when initializing multiple instances of the same procedural
#4805	Autobump crash with NaNs in displacement texture
#4807	Shadow matte broken when there are disabled lights
#4808	AiAOVEnabled crash in free mode
#4809	Light acceleration was missing in free mode
#4811	Inefficient string comparison in deep driver
#4815	Poor sampling quality when tracing camera rays in free mode
#4818	Matte property not working in built-in AOVs (P, N, Z, ...)
#4819	crash due to nodes being initialized multiple times on different threads
#4820	Adaptive subdivision crash if vertex valence exceeds maximum
#4821	Mesh processing corrupts smooth derivatives
#4826	Subdivision crash with user data and free-floating vertices
#4836	Incorrect opacity in utility shader
#4837	Subdivision: exact derivatives not needed for user data and user normals

## 4.2.8.0

### Milestone 4.2.8.0

#### Enhancements

- **Improved volume multiple scattering:** The importance sampling for volume indirect rays has been modified, increasing render time a little, but significantly reducing noise fireflies in various cases. (#4599)
- **Faster .ass writing:** Writing large .ass files is now 40% faster without compression and 8% faster with .gz compression. Limits on the length of string parameter values have been removed from the .ass parser as well. (#4780, #634)
- **Object space adaptive tessellation:** Polymeshes have a new subdiv\_adaptive\_space attribute for subdivision surfaces. subdiv\_adaptive\_space raster corresponds to the previous behaviour and subdiv\_adaptive\_space object corresponds to adaptive subdivision in object space. This is useful for instances or when adaptive tessellation is desirable regardless of a specific camera. In line with this new option, subdiv\_pixel\_error is now called subdiv\_adaptive\_error. Also, Arnold will warn when raster-space tessellation is used on more than one visible instance. (#4758, #4759)
- **Wireframe support for non-linear cameras:** Wireframe renders now preserve line width when using non-linear cameras such as the built-in spherical and fisheye cameras. (#4633)
- **Atmosphere shaders with RGBA output:** Atmosphere shaders can now output alpha through AtShaderGlobals.output.RGBA.a. This is only useful when implementing atmosphere shaders that act as mattes. (#4742)
- **image texture offsets:** The image node now has softset and toffset parameters for offsetting the textures. This offset takes place before scaling, flipping, or swapping of the S and T coordinates. (#4788)
- **Opacity blending supported for FLOAT AOVs:** Opacity blending is now supported for AOVs of type FLOAT, in addition to the previously supported RGB and RGBA. (#4772)
- **Integrator details in logs:** The log files now include more information about the number of samples, GI depths and transparency. Per light volume sampling information was also added. Also, there have been a number of small cleanups and fixes in log files, like removing the legacy, detailed SSS stats, which should result in cleaner, shorter, more reliable and informative logs. (#4721, #4752)
- **Updated to RLM 11.3BL1:** We have upgraded the license server and the external library controlling the licensing subsystem from version 11.2BL2 to 11.3BL1, a more stable release fixing various crashes, bugs, hangs and memory leaks. (#4730)

#### API additions

- **AtString hash:** An AtString.hash() function and AtStringHash functor have been added, which return a precomputed string hash for usage in hash tables like std::unordered\_map. (#4748)
- **Unset message:** AiStateUnsetMsg() functions have been added to remove shader messages from the message bank. (#4747)
- **Set metadata:** Metadata can now be dynamically set on node types by using AiMetaDataSet with an AtNodeEntry. Previously setting metadata was only possible in node\_parameters. (#4766)
- **New procedural plugin methods:** Procedural plugins can now set three additional, optional methods. First, a bounds initialization method allowing the plugin to specify bounds automatically and efficiently for nodes it will generate, rather than relying on the user's specified bounds or on full expansion. Second, plugin initialization and cleanup methods run once for the render session. In order to get access to the plugin-wide data that can be set up during plugin initialization, an additional API function has been provided for the other procedural plugin methods. (#4320, #4769)

```
AI_API void* AiProceduralGetPluginData(const AtNode* );
typedef bool (*AtProcInitPlugin)(void** plugin_user_ptr);
typedef bool (*AtProcCleanupPlugin)(void* plugin_user_ptr);
typedef bool (*AtProcInitBounds)(AtNode* node, AtBBox* bounds, void** user_ptr);
```

#### Incompatible changes

- **Deep EXR files use (AR, AG, AB) as opacity channels:** Deep OpenEXR files were using the old (RA, GA, BA) names for three-channel opacity. We have switched to (AR, AG, AB) to follow the EXR standard. Note that Nuke (as of 9v6) does not yet correctly display these deep files anyway. (#4726)
- **single\_channel texture lookups:** Instead of returning it in the R channel only, the single channel is now returned in the R,G,B channels as monochrome RGB. This affects AiTextureAccess, AiTextureHandleAccess and the image node. (#4741)
- **Default sky.intensity:** The default intensity for the sky background shader has been changed from 0.0 to 1.0. It's unlikely that this will affect old scenes for many users, as this shader was useless with the default intensity. (#4754)
- **Too-small procedural bounds now an error:** Bounds specified for procedurals that end up being too small (and therefore potentially clipping away parts of the object) have been upgraded from a warning to an error. With options.abort\_on\_error turned on, this will halt the render, which is safer and more reliable than random, non-deterministic, hard to debug clipping of these objects. (#4761)
- **Removed AI\_LOG\_SSS:** This bit, part of the AI\_LOG\_ALL bitmask in ai\_msg.h, was unused since SSS pointclouds were removed years ago. (#4794)

## Bug fixes

Ticket	Summary
#4734	Crash during IPR session with Maya on Linux
#3350	AiShaderGlobalsApplyOpacity() not converging on correct result with colored opacity
#4564	Escape sequences in .ass parser not working with string arrays
#4724	Crash with volumes when the P AOV is enabled and an object is behind or inside
#4725	Crash when using deep volumes without an RGBA AOV
#4726	rename "RA" "GA" "BA" OpenEXR opacity channels to "AR" "AG" "AB"
#4728	work scheduler lifetime extension causes crash in yetis
#4729	Rare random buckets rendering too bright
#4731	Missing header info in logs
#4737	Incorrect transformation for lights in motion blurred procedurals
#4741	return monochrome colors from image shader in single_channel mode
#4744	Built-in surface AOVs disappear inside atmosphere shader
#4746	Single channel alpha mismatch in deep driver for volumes
#4750	dPdv on curves is numerically unstable far away from the origin
#4753	BVH motion count in stats is wrong
#4755	fog is not always visible when seen from below the fog plane
#4762	AiShaderGlobals.area does not account for dOdx and dOdy
#4763	Occasional crash when curves generate invalid normals
#4764	crash in subdiv with vertex that exceeds the max valence of 255
#4768	Fix ray differentials in free mode
#4770	AiAOVRegister does not correctly detect unsupported types for AI_AOV_BLEND_OPACITY
#4771	AiMakeRay should compute a more numerically robust AtRay::mindist
#4773	non-deterministic behavior of user data declarations on cached procedural data
#4779	procedural cache wrongly propagates visibility overrides
#4780	Error writing long string parameter values to .ass file
#4792	Crash when re-creating a previously destroyed procedural
#4795	image node with UDIMs with alternate UVs are incorrect
#4798	Random crash with displacement and autobump
#4749	Renaming output in kick fails when a camera is specified in the output
#4765	Python API crash when AtMsgCallback goes out of scope

## 4.2.7.0

### Milestone 4.2.7.0

#### Enhancements

- **Faster cutout opacity mapping:** options.enable\_fast\_opacity can be enabled to get faster textured mapped opacity mask renders, such as for tree leaves. Tests with production scenes have shown up to 25x speedups. This flag also toggles a fix for more accurate renders, where previously the object would be rendered more transparent when further away from the camera. When using this option, noise/flicker can occur on far away geometry, similar to when rendering far away high res geometry. If this happens, it is best to use a high amount of AA samples (and correspondingly lower amount of diffuse/glossy/light samples) to reduce flickering in animation. The fast opacity flag is enabled for anything that is connected to any RGB parameter called "opacity", so custom shaders that have an RGB opacity parameter will benefit from this. We expect that users will want to enable this mode all the time for new scenes and if no one complains, in a future release we will likely remove this flag and leave the optimizations and fixes in the "fast\_opacity" mode always enabled. (#4696)
- **Multiple scattering for volumes:** Indirect light in volumes now supports an arbitrary number of bounces instead of being fixed to one bounce. It is now possible to render volumes such as clouds for which multiple scattering has a large influence on their appearance. The new options.GI\_volume\_depth parameter sets the number of bounces, defaulting to 0. The default value of options.volume\_indirect\_samples has been changed to 2. (#4594, #4682)
- **Support for deep volume output:** Volumes are now visible in deep renders, but note that older "atmosphere" shaders and volumetric mattes are not supported yet. Previously, volumetric samples were composited with the next surface sample. Now, in case you want to implement your own deep driver, volumetric samples are available as independent samples to raw drivers or filters. In order to query the end of a volumetric sample you can request the new built-in float Zback AOV channel. (#4654, #4655)
- **Per light volume contribution:** A volume contribution scaling parameter was added to lights, similar to the existing diffuse and specular parameters. (#4657)
- **New EXR compression modes:** Four additional compression modes, b44, b44a, dwaa and dwab have been added to driver\_exr.compression. b44 is lossy for half data and stores 32-bit data uncompressed. b44a is an extension to b44 where areas of flat color are further compressed. dwaa and dwab correspond to JPEG like compression from DreamWorks Animation. Note that dwaa and dwab require the reading program to be compatible with OpenEXR 2.2 which is not yet widespread, Nuke 9.x will not read them. (#4634, #4638)
- **Faster UDIMs:** UDIMs accessed through the image shader node now internally use texture handles, which helps improve multi-threading performance. (#3058)
- **OIIO improvements:** Upgraded to OIIO to 1.5.15. 16-bit textures are now stored as 16-bit in the texture cache instead of as 32-bit floats. This halves the amount of memory 16-bit textures use. (#4619, #4671)
- **Spaces allowed in EXR metadata names:** driver\_exr.custom\_attributes now supports spaces in names, as in the example below. (#4631)

```
driver_exr
{
    name mydriver
    filename foo.exr
    custom_attributes "POINT2 'Chromaticities/Red Primary' 1.0 2.0"
}
```

#### API additions

- **Per texture cache invalidation:** Sometimes reloading a specific texture is needed, rather than flushing the whole texture cache. AI\_API void AiTextureInvalidate(const char \*filename) is a new API call that lets developers invalidate single textures. (#4698)

#### Incompatible changes

- **Raised minimum OS X platform to 10.8:** OS X 10.7 is no longer supported. You now need at least 10.8 (Mountain Lion) to run Arnold. (#4718)
- **Changed texture\_sss.blur default to 0:** SSS bump textures using sss\_use\_autobump and any textures used in an sss\_irradiance\_shader will now render with more detail. This may lead to increased texture I/O, though we have found little to no impact in various production scenes. (#4664)
- **polymesh.nlist:** Values in this array must be normalized vectors now. (#4651)
- **Removed per-light volume\_density:** This was an old hack that only applied to the volume\_scattering atmosphere shader, and was not supported in the more recent object-based volume API. Note that you can still link volume\_scattering.density to achieve the same effect, only globally instead of per-light. (#4676)
- **volume\_indirect\_samples:** In existing scenes with one bounce of volume indirect lighting, options.GI\_volume\_depth must be set to 1 to re-enable it.

## Bug fixes

Ticket	Summary
#4506	Opacities for volume mattes not as expected
#4596	Crash interrupting render with mesh light
#4606	Crash when calling AiRendering() during AiEnd()
#4624	UDIMs should work when triangles span multiple tiles
#4636	interactive update of light and shadow linking fails randomly
#4644	"too many messages" warning is output for masked out messages
#4645	planar light_blocker does not properly use height_edge
#4652	Procedurals not working correct with _self traceset, self_shadows and receive_shadows
#4656	Overlapping curves objects in procedural sometimes missing in render
#4658	Faceting artifacts with SSS and bump mapping
#4661	Restore shaders (shift + i) in interactive kick not working
#4662	Crash in points primitive with small radii
#4664	excessive texture blur when using sss_use_autobump
#4665	Statistics are not reset for each render session
#4673	Remove non-working light blocker upper case X, Y, Z ramp axes
#4674	INT channels do not work with deep EXR driver
#4682	Volume indirect converges to wrong result due to correlated samples
#4695	clamp away invalid opacity in core shaders
#4697	Improve warnings for varying and indexed data on objects that don't support them
#4699	UDIM textures do not work with UV sets or linked uvs
#4630	silence compiler warning in Visual Studio for AiNodeGetStrAtString
#4659	Fix pedantic warnings in public API

## 4.2.7.1

### Bug Fixes

Ticket	Summary
#4724	Crash with volumes when the P AOV is enabled and an object is behind or inside

## 4.2.7.2

### Bug Fixes

Ticket	Summary
#4725	Crash when using deep volumes without an RGBA AOV

### 4.2.7.3

#### Bug Fixes

Ticket	Summary
#4728	work scheduler lifetime extension causes crash in yetि
#4729	Rare random buckets rendering too bright
#4731	Missing header info in logs

## 4.2.7.4

### Bug Fixes

Ticket	Summary
#4734	Crash during IPR session with Maya on Linux

## 4.2.7.5

### Bug Fixes

Ticket	Summary
#4737	Incorrect transformation for lights in motion blurred procedurals
#4744	Built-in surface AOVs disappear inside atmosphere shader
#4746	Single channel alpha mismatch in deep driver for volumes
#4753	BVH motion count in stats is wrong
#4763	Occasional crash when curves generate invalid normals
#4768	Fix ray differentials in free mode
#4749	Renaming output in kick fails when a camera is specified in the output

## 4.2.6.0

### Milestone 4.2.6.0

#### Enhancements

- **Volume step size in object space:** Previously, volume step sizes were specified in world space, now they are in object space. This means that if you scale down a volume object, the step size will scale down along with it, maintaining the same sampling quality and speed. This affects both the step\_size parameter on nodes, and the automatic step size provided by volume plugins. (#4609)
- **Volume low light threshold:** Volumes now take better advantage of options.low\_light\_threshold, giving around 5% render time reductions in various scenes. (#4610, #4622).
- **Volume stats:** We now report the average number of ray marching samples taken, as well as the average number of volume intersection segments found, per ray type. For shader call stats, "atmosphere" has been renamed to "volume". (#4607)
- **Built-in \_self trace set:** Rays belonging to the built-in \_self trace set will only trace against the current object. This trace set only supports inclusive mode. (#4628)
- **Updated maketx:** Updated the OpenImageIO maketx utility to 1.5.13. (#4591)
- **Searchable SDK:** The Doxygen SDK documentation for the Arnold core API now has a search engine and index for easier navigation. (#3848, #3218)

#### API additions

- **AtString string optimization:** Building off the internal work in 4.2.5, we now expose our AtString class for use in situations where a string is built once and used many times. AtString allows for very fast constant time string comparisons. The tradeoff is that creating an AtString from achar\* string is an expensive operation. For this reason, AtString should be created in a preprocess step, such as node\_update or with a static const AtString and not in performance critical code such as shader\_evaluate. Many API functions that used to have a const char\* as an argument now also accept AtString. Converting your code to use these overloaded functions should improve performance. (#4578, #4603, #4612)

Here is an example of how the MayaRamp shader is made 1.07x faster with the below change to use AtString.

```
AtPoint2 uv = {0.0f, 0.0f};  
- if (!AiStateGetMsgPnt2("maya_ramp_uv_override", &uv))  
+ static const AtString maya_ramp_uv_override("maya_ramp_uv_override");  
+ if (!AiStateGetMsgPnt2(maya_ramp_uv_override, &uv))  
{
```

- **Update method in volume plugin API:** The volume API now provides an update method, much like the node\_update method for nodes. This method is optional and does not need to be provided. If it is provided, it can be used to quickly update volumes during an IPR session, without the volume being created and destroyed every time the scene changes. (#4614)

#### Incompatible changes

- **Raised Linux minimum requirements:** Running Arnold requires a system with at least *glibc* 2.12 and *libstdc++* 3.4.13 (*gcc* 4.4.7). This is equivalent to RHEL/CentOS 6. (#4562)
- **Volume step size in object space:** Due to this change, volume objects with scaling in their transformation will now render with a different effective step size. If this was manually compensated for by changing the step\_size on volume objects or step\_scale on OpenVDB volumes, that must now be undone to avoid too long render times or too low detail. (#4609)
- **Volume shader and sg->P:** When evaluating a volume shader, sg->P can now be located at any point in the volume segment defined by sg->Ro and sg->RI, rather than always being located at the center. We have not encountered any shaders that make this assumption and needed changes, but if any shader does it needs to be updated. (#4587)
- **Tagged unpremultiplied 8-bit TIFFs:** TIFF output images are now marked to have "unassociated alpha" in their header metadata when driver\_tiff.unpremult\_alpha is set to true. Applications that take this tag into account will now display Arnold "unpremultiplied" images correctly. Note that this only affects 8-bit output images. (#4588)
- **Removed shader\_nan\_checks:** The NaN checks after each shader evaluation have been optimized so that they can always be enabled with no degradation in performance. From now on you will always get information about which specific shader node causes a NaN, not just in which pixel. (#2407)
- **Removed obsolete options:** A few global options that were rarely used and never exposed in our DCC plugins have now been removed: error\_color\_bad\_mesh, AA\_motionblur\_pattern, enable\_fast\_lights, enable\_fast\_importance\_tables, mindist\_ulp\_count, bad\_bounds\_slack, max\_shader\_messages. (#4597, #4600)
- **Procedural plugin abort handling:** Previously, if a procedural plugin had started running and the render was aborted or interrupted, the plugin's cleanup callback was not run. Now, the cleanup will always be called once the procedural starts

running even during an abort. Not all nodes may have been requested yet, however, so the plugin writer should keep in mind that the AtProcCleanup callback may get invoked before the AtProcGetNode callback has been invoked as many times as expected. (#4625)

- **Restore previous signal handlers when crashing:** Arnold used to replace the preexisting signal handlers with its own handlers while it was running and after running it set them to the default signal handlers. If the parent process (e.g. Houdini) had set its own signal handlers, these handlers were permanently lost. Now we restore these handlers when Arnold finishes rendering. (#4621, #4629)

## Bug fixes

Ticket	Summary
#4595	XSI/C4D hang on exit after rendering
#4449	Bump evaluation can request non existing UDIM tiles
#4552	crash when setting AOVs in background or atmosphere shaders
#4577	EXR driver append mode broken for some image resolutions
#4585	Crash with procedural DSO in second Arnold session
#4587	Volume artifacts at large step sizes
#4598	NaNs in alpha channel of float image textures without alpha
#4601	overrides on options not working
#4602	Matte objects still set AOVs when output type is RGB instead of RGBA
#4615	uninitialized shaderglobals privateinfo are common problems
#4621	restore previous signal handlers when arnold uninstalls its signal handlers
#4625	Render abort/interrupt leaves procedurals no chance to clean up
#4629	Pass signals from a crash to parent process

## 4.2.4.0

### Milestone 4.2.4.0

#### Enhancements

- **Transmission component in hair shader:** The built-in hair shader now includes an additional energy conserving, Marschner-style transmission component (also known as TT). Its angular size can be controlled with hair.transmission\_spread which acts as a multiplier on the values for blonde hair: a smaller spread will correspond to more focused brighter effect. Default spread is 1.0, and reasonable values are in the 0.5 to 5.0 range. Like hair.spec and hair.spec2, hair.transmission can be turned on and off per light with light.affect\_specular. (#4492, #4508)
- **Volume mattes:** The matte parameter is now respected for volume objects, where they will skip lighting and render faster, with the total opacity of the volume cutting out alpha just as surface matte objects do. (#4476)
- **Extended motion vector support:** Motion vectors are now correctly computed for all built-in cameras, not just persp\_camera. (#4473)
- **Extended min\_pixel\_width support:** Minimum pixel width on points or curves now works for all built-in cameras, not just persp\_camera. (#4478)
- **Support node-locked licences:** Node-locked licenses can now be automatically loaded if they are placed along with the Arnold binaries (libai.so, ai.dll or libai.dylib) in the Arnold installation folder. If placed in other location, it should be pointed by the solidangle\_LICENSE environment variable, specifying the path. Both, RLM servers (in the port@host form) and paths (to folders containing license files, or full paths to specific license files) can be specified at the same time, separated by : or ;, depending on the platform. We encourage to read the [RLM License Administration documentation](#). (#4501)
- **Updated to RLM 11.2BL2:** We have upgraded the license server and the external library controlling the licensing subsystem from version 11.1BL2 to 11.2BL2, a more stable release fixing various crashes, bugs, hangs and memory leaks. (#4375)

#### API additions

#### Incompatible changes

- **Gamma for 8-bit RGBA outputs:** For 4-channel RGBA output files, gamma correction is no longer applied to the theoretical "unpremultiplied value" but to the raw RGB value. This increases quantization accuracy and works better for volumes or transparent surfaces. Because of this change, the Nuke Read node will work with its default settings, without the need to check premultiplied. (#4504)
- **Fog shader opacity:** Previously, the atmospheric fog shader would occlude, but would not appear in alpha, nor in the opacity AOV. The fog shader will now show up in both cases. (#4519)
- **Atmosphere shaders and sg->Vo:** This is a reminder that custom atmospheric shaders (those attached to options.atmosphere) are now required to fill out sg->Vo with the volume's additive color in the same way that volume shaders already do. For instance, for a fog shader whose atmosphere contribution scatters fog\_radiance towards the ray origin and attenuates the background according to fog\_opacity, things should be set in a similar way to this: (#4519)

```
sg->Vo = fog_radiance;
sg->out.RGB = sg->Ci * (AI_RGB_WHITE - fog_opacity) + fog_radiance;
```
- **AiLicense{Get|Set}Server() and ARNOLD\_LICENSE\_{HOST|PORT} have no real effect:** Arnold will no longer listen to the deprecated ARNOLD\_LICENSE\_{HOST|PORT}environment variables. In order to specify a list of license servers or license files, you must use the standard RLM environment variables solidangle\_LICENSE or RLM\_LICENSE. The AiLicense{Get|Set}Server() API is now useless as it does nothing and always returns false, and will be removed in a future release. Starting with this version, Arnold will automatically look for license files in the installation folder and it will also broadcast license servers in the LAN. (#4059)
- **Some warnings upgraded to errors:** We have made warnings and errors more consistent regarding geometry nodes such as curves, polymeshes, points, etc. As a result, particularly curves nodes will emit errors when certain parameters are missing or are invalidly constructed, which if options.abort\_on\_error is on will terminate the render immediately. It is usually better to abort the render and require the data to be fixed instead of rendering more quietly but incorrectly, so we have made this policy more consistent for these nodes. (#4513)

#### Bug fixes

Ticket	Summary
#4463	Crash with AiTraceProbe and curves
#4470	instance memory overhead under-reported
#4472	acceleration structure for points over-reporting memory used
#4484	Tag volumes without a valid shader with options.error_color_bad_shader
#4487	Incorrect volume shading between a transparent surface and the background
#4494	texture lookups with ignore_textures enabled should count as a successful texture lookup
#4503	increase AI_MAX_PARAM_OVERRIDES
#4504	Do not divide by alpha ("unpremultiply") when applying Gamma on 8bit RGBA images
#4505	rare crash loading binary encoded .ass files
#4510	Don't crash when shidxs array has incorrect size
#4513	Improve warnings/errors for shape nodes
#4519	Atmosphere shader evaluation should only use Vo and out.RGB
#4526	kick should match data type and filter when displaying the RGBA AOV
#4527	Artifacts with overlapping oriented curves
#4502	AiSamplerSeeded() generating correlated samples with nsamples 1

## 4.2.3.0

### Milestone 4.2.3.0

#### Enhancements

- **Faster curves:** Several optimizations have been done on the ray/curve primitive test giving up to 40% faster hair performance. In the process, the max\_subdivs attribute of the curves node has been removed. In addition, the internals of the curves primitive acceleration structure have been tweaked, giving up to 20% memory savings with 10% faster preprocessing. (#4386, #4411, #4422, #4433, #4437, #4441, #4454)
- **Faster volumes:** Shadow rays through both bounded and atmospheric volumes are now faster. If image artifacts are noticed, decrease the volume's step\_size. Speedups are scene-dependent, but we've seen more than a 2x speedup in many of our tests, with little to no quality degradation. (#4377, #4383, #4429)
- **Up to 128 threads:** Arnold now allows for rendering with up to 128 threads (increased from 64). Shaders that make use of AI\_MAX\_THREADS will need to be recompiled otherwise crashes may occur in upcoming machines with more than 64 threads. If you do not have machines with more than 64 threads, there is no need to recompile any shaders. In addition, like in Linux, Windows can now make use of the existing options.pin\_threads. (#4337, #4330)
- **Parallel node initialization/update:** Initialization and update of scene nodes can now be multi-threaded. This can significantly lower the scene preparation time in complex scenes with many objects, shaders or lights. This extends the parallel procedural initialization implemented in the previous 4.2.2 release. The previous parallel\_procedural\_init option has been renamed to parallel\_node\_init because it now applies to all nodes, not just procedural nodes. For now, this option is disabled by default, to avoid issues with existing custom shaders or procedurals that may have hidden implicit dependencies between nodes, something which is not currently compatible with multi-threading. Please try this out and report any problems. (#4329, #4358)
- **Upgrade to latest compiler:** OS X and Linux use a newer version of clang. This results in slightly faster code. Shaders and plugins can safely continue to be built with older compilers. (#4354, #4356)
- **User normals in linear subdiv and displacement:** User-specified polymesh vertex normals are now preserved and respected during displacement, and also when using linear subdivision. For cases where Catmull-Clark subdivision is not needed, this allows sharp edges to remain sharp when increasing mesh density for detailed displacement. Catmull-Clark subdivision still warns about the presence of user normals and throws them away in favor of the smoothed limit normals. It is still usually preferable to use Catmull-Clark subdivision with the new fractional creasing instead of linear subdiv and user normals, however. (#4306)
- **Per-node 'active' flag:** Scene nodes can now be enabled or disabled for rendering (this is a debug feature that is not preserved in .ass files). For now only lights, shaders and objects can be enabled or disabled. (#4150)
- **Kick display of individual color channels:** Pressing shift + r, g, b, a in interactive kick now displays the corresponding color channels. shift + c resets to RGBA display. (#4366)
- **Autobump support for meshes with no UVs:** The autobump detail enhancement now works for procedurally displaced polymeshes even when there are no UV texture coordinates. (#4453)

#### API additions

- **Per-node 'active' flag:** New API functions are provided to set and query the new flag: (#4150)

```
void AiNodeSetDisabled(AtNode* node, bool disabled);
    bool AiNodeIsDisabled(AtNode* node);
```

#### Incompatible changes

- **Raised minimum Windows version to Windows 7:** We no longer support Windows versions before Windows 7 and Windows Server 2008 R2. (#2871, #4330)
- **Atmosphere shaders and sg->Vo:** Custom atmospheric shaders (those attached to options.atmosphere) are now required to fill out sg->Vo with the volume's additive color in the same way that volume shaders already do. For instance, for a fog shader whose atmosphere contribution scatters fog\_radiance towards the ray origin and attenuates the background according to fog\_opacity, things should be set in a similar way to this: (#4389)

```
sg->Vo = fog_radiance;
    sg->out.RGB = sg->Ci * (AI_RGB_WHITE - fog_opacity) + fog_radiance;
```

- **AiTrace() for shadow rays:** If AiTrace() is passed a ray of type AI\_RAY\_SHADOW, it will now more quickly compute the sample opacity, but it will no longer return any other sample information. If that is required, AiTraceProbe() can be used. Note that, previously, the sample information it provided was not for the first hit, but rather a random hit. So aside from the opacity, this never provided reliable sample information. (#4377)
- **Full-frame spherical and cylindrical cameras:** Spherical and cylindrical cameras now maintain their FOV vertically regardless of frame and pixel aspect ratio. For spherical cameras this means that the render will always reach all the way to the poles vertically.

For cylindrical cameras vertical\_fov will now be correctly respected. Tweaking options.aspect\_ratio is not needed anymore to get the correct FOV. This change does not affect custom cameras. (#4398)

- **Removed enable\_aov\_composition:** The global option enable\_aov\_composition no longer had any effect on performance so it has been removed and now compositing of semi-transparent AOVs is always performed correctly. (#4404)
- **RGBA AOV composition:** The A channel of RGBA AOVs is now fully composed as an additional alpha channel per AOV. Previously no compositing would happen for this channel, and just the value from the first surface hit would be reported (irrespective of the particular AOV being set or not) (#4408)
- **8-bit output when alpha is 0:** A bug has been fixed that was preventing correct file output if A was set to 0 and gamma correction was used. Before this bug was fixed, the output would have been (0,0,0,0) instead of the correct (R, G, B, 0). (#4420)
- **AI\_MAX\_THREADS increased to 128:** AI\_MAX\_THREADS has been increased to 128, so any code that relies on it and needs to support more than 64 threads must be recompiled. (#4337)
- **Removed legacy cubic curve support:** The unused cubic bases hermite and power in the curves node have been removed. (#4441)
- **Removed curves.max\_subdivs:** The max\_subdivs attribute of the curves node has been hardcoded to 6 (the previous default) and removed. The only reason to tweak this parameter was to get a tiny speedup at the cost of lower quality, faceted curves; this tiny speedup is now dwarfed by the more significant ray/curves speed gains found in this release.
- **Autobump enabled for meshes with no UVs:** Previously polymesh.disp\_autobump true was ignored on polymeshes without UV texture coordinates. Now, autobump will be correctly applied which might cause a change in look (improved displacement detail) for that specific case. (#4453)

## Bug fixes

Ticket	Summary
#4393	main thread was pinned to a single core
#2871	Windows CPU detection code is wrong for high-end machines
#4330	Windows is unable to use more than 64 logical cores
#4355	Compute correct differentials when persp_camera.uv_remap is used
#4359	memory leak in AiTraceProbe()
#4367	Matte override on ginstance is ignored
#4378	streak artifact in quad lights
#4385	Kick display issue with fast cropped renders
#4387	excessive volumetric noise in fast motion-blurred quad lights
#4390	Skydome light crash and IPR issues with null energy in linked color
#4394	Kick display window issues with old or incomplete OpenGL drivers
#4398	Spherical and cylindrical cameras incorrect cropping for non square frames
#4402	User data of type array crashes when accessed for displacement
#4403	Kick displays a black window when returning from the lock screen
#4408	RGBA AOV alpha component should composite using the global alpha
#4409	Bright dots in fog at the horizon line
#4410	random flat areas in bump mapping
#4420	Unable to output purely additive pixels with gamma
#4428	Polymeshes with zero area faces can have wrong user data for some faces
#4430	Curves incorrectly clipped in render in some cases
#4442	Allow for motion blurred FOV on custom cameras tagged with is_perspective
#4448	Compositing issues with fully transparent samples
#4452	Bad normals with autobump and vector displacement
#4457	Crash with invalid time_samples parameter
#4439	pykick doesn't understand negative AA samples on command line
#4462	Metadata files with Windows EOL are not correctly parsed in OSX

#### **4.2.3.1**

##### **Milestone 4.2.3.1**

Ticket	Summary	Keywords
#4463	Crash with AiTraceProbe and curves	4.2.3.1

## 4.2.2.0

### Enhancements

- **Subdivision fractional soft creases and vertex creases:** OpenSubdiv-compatible fractional soft creases are now supported through a new polymesh.crease\_sharpness attribute that holds the sharpness value for each crease listed in polymesh.crease\_idxs. A crease sharpness value bigger than polymesh.subdiv\_iterations will de facto encode an infinitely sharp crease. If crease\_sharpness is not present, all creases will be assumed to be infinitely sharp. Additionally, it is possible to set vertex sharpness by duplicating the vertex in crease\_idxs. For example:

```
polymesh
{
    crease_idxs 1 2 3 4 5 5
    crease_sharpness 1e6 2.5 4.0
    ...
}
```

marks the edge between vertices 1 and 2 as infinitely sharp, the edge between vertices 3 and 4 as having sharpness 2.5, and sets the sharpness to 4.0 for vertex 5. (#3736)

- **Multi-threaded loading of pre-expanded procedurals:** It is now possible to load multiple procedural/standin nodes in parallel at node initialization time, before ray tracing begins, i.e. when either procedural.load\_at\_init or options.procedural\_force\_expand are enabled. For big scenes that rely heavily on procedurals, this can dramatically speed up scene loading. This is an experimental feature, and as such is disabled by default. Procedurals that are not thread-safe, or procedurals that depend on other nodes having been previously initialized, could result in crashes and undefined behaviour. This feature can be enabled with the new option parallel\_procedural\_init. We might remove or rename this option based on production feedback. (#2957, #4280)
- **Multi-threaded importance tables in textured quad\_light:** The importance sampling tables needed by textured quad area lights can now be computed in parallel. This can result in important speedups in scenes with many lights that use high-resolution tables. (#4227, #4280)
- **Faster thread creation/destruction:** The overhead when creating and destroying rendering threads has been reduced. This is specially noticeable when using interactive rendering with 32 or more threads. (#4188)
- **Adjustable thread priority in OS X:** OS X now allows for options.thread\_priority to adjust the priority of the render threads. It defaults to lowest, which is lower than the system default that was previously being used. Previously this option only existed in Windows. (#4292)
- **Reduced texture IO:** Texture IO is lower for texture lookups seen through glossy reflections and for displacement textures. This can result in noticeable speedups in texture-heavy scenes. (#4311)
- **Faster kick display:** The -interactive mode renders faster and navigation is smoother with less flickering. OpenGL graphics drivers are now required to open the kick render window, but batch rendering without a window does not require OpenGL to be installed. On Mac OS X, it is no longer required to install X11 to run kick. (#3063, #4210, #4304).
- **Faster SSS sets:** Sub-surface scattering is now faster for rendering overlapping hair and skin together if the skin objects were included in an SSS set. As an added bonus, this fixes long-standing darkening artifacts when using SSS sets near dense hair areas. (#2988, #3906, #4324)
- **More robust UDIM tile handling:** A couple of new attributes have been added to the image node. By default, missing UDIM tiles will still generate an error. However, certain workflows require missing tiles to be ignored and replaced with a default texture. This can now be done by setting image.ignore\_missing\_tiles true and linking or setting the image.missing\_tile\_color attribute to a constant color. Additionally, the code is now more robust to specific illegal UV mappings resulting in negative indices near polygon edges. (#4346, #4332)
- **Removed image.cache\_texture\_handles:** The image shader now always uses fast texture handles under the hood, and this can no longer be manually turned off. The only case where texture handles are not used is if the filename string is linked, but we don't recommend this as multi-threading performance would suffer. (#4351)
- **Improved matte/holdout support:** All objects now have a matte boolean parameter to turn them into a holdout. Shaders will not run on the object anymore and it will output all-black (including the alpha), except if the opaque parameter is off in which case shaders will be run just to compute the opacity. Note that even AOVs output by its shaders in that case will be black. (#4299)
- **Volume instancing:** The volume node didn't forbid instancing previously, but didn't work. We now formally support instancing of volume nodes via ginstance nodes as usual. (#4277)
- **Self-only mode for occlusion:** The ambient\_occlusion shader makes use of the new AiSelfOcclusion() API function when requested to only gather occlusion against the same object. (#4285)
- **Spiral buckets by default:** The default options.bucket\_scanning has changed from top to spiral, a more IPR-friendly mode. If rendering to scanline-based EXRs in batch mode on the render farm, the top mode might still be a better option. (#4274)
- **AiUniverseCacheFlush(AI\_CACHE\_TEXTURE) always works:** Textures can now be flushed during rendering. If a texture flush is requested during rendering, Arnold will record this request and when it finishes rendering, the texture flush request will be passed on to the texture system. Generally, a texture flush should be paired with an AiRenderInterrupt() so that the renderer can immediately stop and flush the textures. (#4341)
- **OIO improvements:** OpenImageIO has been upgraded to 1.4.14. Texture lookup performance under many threads has been improved when performing many invalid texture lookups, for instance, when performing displacement mapping with an invalid displacement texture. Fixed a bug in OIO which was causing random crashes in Windows during texture lookups. (#4202, #4282, #4296, #4312)
- **Updated to RLM 11.1BL2:** We have upgraded the license server and the external library controlling the licensing subsystem from version 10.1BL2 to 11.1BL2, a more stable release fixing various crashes, bugs, hangs and memory leaks. (#4088, #4258)

## API additions

- **Self-only occlusion:** AiSelfOcclusion() was added to allow intersecting only against the same object which is being shaded. It takes the same parameters as AiOcclusion() and can be used as a drop-in replacement for those situations where this type of occlusion query is needed. (#4285)
- **Matte status:** AiShaderGlobalsIsObjectMatte() is now provided to quickly query the new matte parameter that objects have. This can be used in conjunction with rays of type AI\_RAY\_CAMERA to only compute opacity, and skip all other shader logic. (#4299)
- **Subsurface ray type:** AI\_RAY\_SUBSURFACE is a new internal ray type used for BSSRDF and single scattering probe rays. The ray counts section of the log includes these ray types as bssrdf and single\_scatter. This internal ray type does not currently affect the object visibilityparameter. (#4324)
- **User parameter in shaders:** Shaders may now get a AtUserParamEntry\* for a given user parameter via AiUDataGetParameter(const char\*). This allows the shader to inspect various aspects of the user data, including the type, so that the appropriate AiUDataGet\*() call may be issued. Retrieving the parameter will return NULL if the parameter doesn't exist as user data. (#4342)

## Incompatible changes

- **SSS with zero diffusion radius:** previously such shaders would render black, now the result is the same as a diffuse BRDF. This affects the standard shader, as well as the AiBSSRDFCubic() and AiBSSRDFGaussian() functions. (#3272)
- **Sharper ray derivatives:** The directional derivatives (dDdx and dDdy) are now based off the distance between samples (clamped by texture\_max\_sharpen) and not the distance between pixels. The first implication of this is that texture\_max\_sharpen does not affect texture lookups for displacement and other places where the ray derivatives are not used in computing the texture derivatives. The second implication is that ray derivatives can no longer be directly used to compute pixel footprints. To do that, the derivatives would need to be widened by:

```
if (options->AA_samples > 1)
    my_derivative *= MIN(options->texture_max_sharpen, static_cast<float>(options->AA_samples));
```

However, beware that this could result in over widening for reflected rays. (#4311)

## Bug fixes

Ticket	Summary
#4260	Overscan output invalid for non-tiled EXRs
#3272	correctly handle tiny/zero diffusion radius in SSS
#4202	soft invalidations do not work for relative paths
#4233	holes in the alpha channel when shading layers at auto_transparency_depth
#4243	Overriding per-face shaders in ginstance gets wrong indices
#4263	black dots/nans near center of spotlight with cosine_power > 0
#4265	8 and 16 bit gamma corrected textures generate some values above 1
#4267	rare random crash at start of rendering
#4269	Perlin noise returns NaNs for very large inputs
#4271	INT and BOOL AOV filtering not working in some cases
#4272	crash after deleting a shader node connected to "background" or "atmosphere"
#4275	Subdivision crash due to unsupported vertex valence (bigger than 255)
#4277	Volume crash when used with instancing
#4281	Curves UV discontinuity in linear mode
#4287	AiSamplerSeeded() not fully using seed so still has some correlation
#4288	AiBegin should check if it's already in a session
#4294	Remove utility.set_opacity
#4297	Potential race condition when creating internal shaders
#4308	Quad light artifacts with small or far away lights
#4309	Camera depth of field and other parameters not updating correctly in IPR
#4311	displacement texture derivatives should not be affected by AA samples
#4312	oiio crashes in windows
#4313	Volume shader override not working
#4314	Normals around hard subdiv creases are smoothed (non adaptive mode)
#4321	simple primitives don't update/rebuild properly
#4326	AiNodeResetParameter inheritance issues and missing python bindings
#4332	UDIMs and negative UVs
#4343	min_pixel_width fails for points and curves created in order before a camera
#4344	min_pixel_width silently fails for points nodes with a single point
#4349	varying string user data crashes arnold
#4325	remove extra newline in .ass file array declaration of strings
#4353	image multiply/offset should have 'linkable' metadata enabled

## 4.2.0.0

### Milestone 4.2

#### Enhancements

- **Stereo/multicam rendering:** The output statements have been extended to accept an optional camera name as the first parameter in an outputs string. Custom exporters or host application plugins can use this to setup multi-camera renders. The new additional syntax is:

```
"camera AOV_name AOV_type filter driver" # for a regular driver  
"camera driver" # for a raw driver
```

Multi-camera renders of the same scene can share the reading in and processing of files, acceleration structures, tessellation, displacement and texture cache by having the different cameras render sequentially. Interleaved bucket rendering for different cameras or sending the results from multiple cameras to the same multi-layer driver is not yet supported. (#2404)

- **Improved motion blur sampling quality:** Motion blur sampling has been improved, resulting in less noise given the same AA sample and time budget. The improvement is most prominent in scenes where motion blur is the primary source of noise in the image. There are now two possible values for the AA\_motionblur\_pattern option: the new default pattern, and the legacy jittered pattern which is provided as a backup until the new sampler is sufficiently battle-tested in production. (#2131, #4037, #4044, #4099)
- **Improved glossy sampling quality:** Some scenes will now render with slightly less noise given the same amount of samples and time. In particular, microfacet-based BRDFs viewed at grazing angles will show significant improvement. (#3571, #3888, #3900)
- **Improved CPU utilization:** The thread scheduler has been redesigned achieving much better CPU utilization in scenes that get stuck in a particularly slow bucket near the end of the frame. It is no longer necessary to use very small buckets to prevent these slowdowns. (#3873)
- **Improved memory performance and usage in Linux:** Linux-specific optimizations have been added which reduce memory usage and improve the threading scalability in certain situations, usually during scene preprocessing and shutdown. These require matched AiMalloc() / AiFree() allocations. If you allocate memory in your shader/procedural with AiMalloc() and then free it with free(), it is possible Arnold could crash. (#3779)
- **Faster ray accel build:** Acceleration structure construction is now up to 1.3x faster while using the same memory. (#3898)
- **Faster subdivision:** Catmull-Clark mesh subdivision is now up to 1.5x faster and uses about 5% less peak memory. (#3903, #3913, #3949)
- **Faster displacement:** Polymesh vertex displacement has been sped up, particularly for lower thread counts. On some highly subdivided models we've seen around 2x faster displacement even with 8 threads. In addition, vector displacement artifacts on UV seams have been eliminated. (#4008, #4017)
- **Faster polymesh processing:** Polymesh sanity checks are now about 3x faster, so we have removed the seldomly used options.mesh\_sanity setting and now always perform these checks. Mesh processing/optimization time is now reported in the stats and this step is about 1.5x faster than before. (#3912, #3917, #3927, #3952, #3986)
- **Faster rendering of many-light scenes:** Scenes with many spaced out lights will now render faster thanks to improved culling of low contribution lights. We've seen up to 2x faster renders in some challenging production scenes. (#4100)
- **Faster SSS:** Raytraced diffusion-based SSS should now show higher performance on objects that have not been assigned to be a part of an SSS set via a "sss\_setname" user data string, specially on scenes with spatially overlapping objects, like hair over skin. (#3808)
- **Faster image shader:** Texture map lookups coming from the image shader are now significantly faster. The noise shader has been made slightly faster too. (#3963, #4114)
- **Faster AiNodeGet\* and motion vectors:** AiNodeGet\*() is now substantially faster on multi-threaded machines. Before, excessively calling these functions would cause Arnold to not scale to multiple threads. Now Arnold is able to scale when these functions are called during rendering. There is still a substantial overhead for calling these functions and so AiNodeGet\*() usage is still discouraged from being called during shader evaluate. Usage of AiWorldToScreenMatrix(), AiShaderGlobalsGetPixelMotionVector(), and AiDriverGetMatrices() are all substantially faster now and will scale to multiple threads. One benefit of this is that computing motion vectors is now substantially faster. (#3096, #3836, #3849)
- **Faster string processing:** We now scale better when handling strings. For instance, if many threads are frequently making calls that pass in strings, such as AiNodeDeclare(mynode, "some\_string", "uniform RGB") we will now scale to all these threads and execute this call more quickly. This matters mostly in deferred loading of geometry from DSO/.ass procedural nodes. (#3921, #3957)
- **Faster AiNode():** Performance scaling has been improved when many threads are rapidly calling AiNode(), as it can happen with deferred loading of geometry from DSO/.ass procedural nodes. Under normal situations, this bottleneck is not hit and so no speedup will be seen. (#3954)
- **Faster deep output using less memory:** Renders that output deep images are now up to 15% faster and use less than half the memory. (#4081, #4102)
- **Faster AOV composition with no memory overhead:** The memory overhead incurred when usingoptions.enable\_aov\_composition has been eliminated. This can save gigabytes in highly-threaded, high AA, multi-AOV renders. Complex renders with lots of transparency and AOVs could be up to 20% faster. (#4038, #4065)
- **Faster image output:** Taking advantage of the non-locking process\_bucket API call introduced in 4.1, we have increased concurrency and improved the speed at which AOVs are written to disk, with a 15% to 30% speedup in built-in drivers

like driver\_exr, driver\_tiff, etc. Custom drivers written by third-party developers should be updated accordingly for maximum performance. (#3857)

- **Faster backlighting:** The translucency, or "backlighting" computation in the standard shader, controlled with the Kb parameter, will use on average 50% fewer shadow rays. (#3890)
- **Skip skydome importance table if color is not linked:** Skydome lights will no longer build their importance tables when the color is not linked to anything. This can save a few seconds of preprocessing time when using high resolution tables. Also, IPR renders will now automatically respond to changing the skydome color. (#3020)
- **autobump in SSS:** Arnold can now optionally take into account the effect that autobump has on the ray traced BSSRDF's result via the `ssss_use_autobump` render option. This helps capture the high frequency details of the surface more accurately, at the expense of slightly longer render times. For backwards compatibility and performance reasons, this option defaults to false. (#3872, #3894)
- **Smoothen bump near shadow terminator:** We have fixed long-standing faceting issues near the shadow terminator boundary when using strong bump mapping. The improvement is most noticeable with detailed, high-frequency maps or when seen from far away. (#3891)
- **Reduced edge darkening in microfacet BSDFs:** Thanks to a new less aggressive shadowing/masking term, the built-in Cook-Torrance BRDF and microfacet BTDF now conserve a bit more energy and exhibit less edge darkening, particularly at high roughness settings. The impact is expected to be very subtle in most scenes. (#4043)
- **Anisotropic Cook-Torrance BRDF:** The Cook-Torrance BRDF functions in the shading API as well as the standard shader now actually make use of the anisotropic shading parameters. This BRDF has an anisotropic shading effect that is very similar to the anisotropy in the Ward-Duer BRDF in behavior, but with a greater amount of energy conservation and less edge darkening. (#4042)
- **Improved ambient and reflection occlusion:** Ambient occlusion artifacts around edges of low-poly objects have been cleaned up. In addition, the object visibility attributes (`visibility`, `self_shadows`) or the minimum occlusion distance (`mint > 0` in the `AiOcclusion` call itself) can now be used to remove artifacts near silhouettes for reflection occlusion, or to disable occlusion from bump-mapped normals. (#3972)
- **Ambocc distance in utility shader:** Added a new `ao_distance` parameter in the utility shader to control the length of ambocc rays for scenes where the default of 100 units is too short. (#3861)
- **Refraction opacity AOV:** The standard shader now can output refraction opacity as an AOV. It is advised to write the complete RGBA set of channels, since the alpha component will constitute a mask that can be used to mix it in with the regular opacity, multiplying in refraction opacity to regular opacity based on the mask. (#2880)
- **Motion-blurred rolling shutter:** Cameras now have a `rolling_shutter_duration` parameter with which it is possible to control the duration of exposure of the scanlines in a rolling shutter camera. Valid values for this parameter are in the 0 to 1 range, where a value of 0 gives you an instantaneous exposure of each scanline (the default value and the rolling shutter's previous behavior), and a value of 1 exposes every scanline for the entirety of the camera's shutter interval (the same result that a camera without rolling shutter would give). (#4009)
- **Support for custom shutter shapes:** Added an additional curve value to the `shutter_type` enum in cameras. This, in conjunction with a new camera parameter `shutter_curve`, allows arbitrary shutter shapes, linearly interpolating shutter open/closed values between points. You can define as many points as needed. Coordinates are increasing in X from 0 (corresponding to `shutter_start`) to 1 (corresponding to `shutter_end`), and values in Y must be non-negative. (#3934)

```
# example shutter_curve usage
persp_camera
{
    ...
    # double-trapezoid shutter with outer edges being more gentle,
    # and the first trapezoid emphasized much more
    shutter_type curve
    shutter_curve 8 1 POINT2
        0.0 0.0
        0.2 3.0
        0.3 3.0
        0.4 0.0
        0.6 0.0
        0.7 1.0
        0.8 1.0
        1.0 0.0
}
```

- **Volume plugins:** A new node volume has been added which allows volume plugin DSOs (specified via the `dso` parameter of a volume node). Volume plugins allow a plugin writer to wrap any volume format or generator for efficient rendering. Volume plugins provide volume creation, destruction, channel sampling, and gathering of ray extents, as well as an automatic bounding box and step size for ray marching (although these can be overridden by the user in the volume node). Ray extent computations inside the plugin will tightly bound volume data along the ray so that the number of volume shader evaluations are drastically reduced in sparse volume datasets. For a short tutorial on how to implement your own volume plugin DSO, see this [article](#) on the arnoldpedia. (#3466, #3978, #3996, #4045, #4046)
- **Density shader:** A new built-in shader density is available which uses `AiVolumeSampleChannel()` to sample from the new volume plugins. This allows volume sampling to work regardless of what source the volume data comes in (as long as there is a volume plugin for the source). Currently it has parameters for controlling RGB scattering, absorption, emission, forward vs backward scattering strength, interpolation quality, and offsetting the sampling position (for example with a noise shader to create the appearance of a higher resolution volume than what came from the source data). (#3466, #3994)
- **Static OIIO linking in Windows:** Just like we do in Linux and OSX, we now statically link the OpenImageIO library in Windows too; we don't distribute the file `OpenImageIO.dll` anymore. (#3969)

- **Scene update on "free" mode:** When using the "free" render mode, you can now force a scene update using AiRender(AI\_RENDER\_MODE\_FREE), which will take into account parameter changes and nodes created or destroyed. (#3862)
- **Multithreading in procedurals:** The enable\_threaded\_procedurals option no longer affects the loading of the text-based .ass, .obj and .plyformats, which are now always loaded in parallel. This option will still apply to binary procedurals (.so/.dll/.dylib), forcing sequential processing when disabled. (#4096)
- **IES files in texture\_searchpath:** If not found in the default path, photometric IES files are now searched for in texture\_searchpath too. (#4036)

## API additions

- **is\_perspective metadata:** View-dependent geometry effects like min\_pixel\_width and adaptive subdivision are now supported in custom camera plugins that have a float fov parameter. This requires the developer of the custom camera to set the boolean is\_perspective metadata to true. An example is given below. NOTE: we reserve the right to supersede this with a better API at some point in the future. (#2756)

```
node_parameters
{
    AiParameterFlt("fov", 60.0);
    AiMetaDataSetBool(mds, NULL, "is_perspective", true);
}
```

- **AiPoint() and AiVector():** A new API function AiPoint() has been added which is aimed at replacing the old and deprecated API AiV3Create(). An equivalent function AiVector() has been added, simply for naming consistency. Analogously, 2D functions AiPoint2() and AiVector2() have also been added. These new APIs allow writing shorter, more concise code in certain cases. (#3936)
- **Volume plugin API:** Volume plugins provide callbacks very similar to procedural plugins. The callbacks are for initialization and takedown of the plugin, initialization and destruction of volumes, sampling of volumes, and providing tight ray extents along a ray where there is volumetric data to sample (via AiVolumeAddIntersection()). Volume plugins may query declared user parameters from the volume node to customize their loading of volume data as needed. Volume plugins are expected to provide a bounding box around the volume data, as well as a recommended step size for ray marching the data. Shaders may query the volume plugin for data on specific named channels via AiVolumeSampleChannel(). Please see the API documentation for details and this short tutorial [article](#). (#3466)
- **External memory tracking:** A new API function AI\_API void AiAddMemUsage(AtInt64 size, const char\* category) is available for plugin writers and API users to inform Arnold when they allocate and deallocate memory. The string category used will show up in log files under the memory stats at the end of each AiRender() call. It is strongly advised that all shaders, plugins and procedurals that need to allocate memory make use of this new API so that log files can tell the true story of where memory is being used, otherwise it all goes into the "unaccounted" line of the memory stats. (#3948)
- **Procedural with variable number of nodes:** Procedural nodes are now allowed to return 0 from the NumNodes() callback, to avoid having to return a fixed number of nodes when that might not be known in advance. In this case, the GetNode() callback will be called in sequence until it returns NULL. (#3971)
- **Reset parameter to default value:** A new function AiNodeResetParameter() is provided to reset a node parameter to its default value, also removing any links to that parameter. (#4071)

## Incompatible changes

- **Empty sss\_setname strings ignored:** Objects with SSS that have been assigned to be part of an SSS set via an empty "sss\_setname" user data string will no longer be considered as part of the same set, which may result in differences in the rendered result. (#3923)
- **Removed cubic projection map:** This old projection map is rarely used, has seam artifacts along cube edges, and is the least efficient of the various possible cubic formats, so we have decided to remove its support in the sky and skylight nodes. Note that we have left the C API intact, AiMappingCubicMap() and AiMappingCubicMapDerivs(), in order not to break binary compatibility in the rare event that anybody was using these API calls. These calls are now marked as deprecated and will be removed in a future release. (#3937)
- **edgelength changes:** The edge\_length color mode in the utility shader now works on any mesh, regardless of whether it's subdivided or not. This visualization is handy for detecting over-tessellated objects. The old behaviour has been moved to a new mode called pixelerror, which is similar but based on how well the polygon matches subdiv\_pixel\_error. (#3956)
- **Early abort when no outputs are present:** When no output drivers are present the render will now be canceled. This has the side effect that progressive kick renders will be aborted when no display driver is present. If for debugging reasons you wish a render to continue even without valid outputs you can use options.abort\_on\_error false. (#3668)
- **Deep EXR driver tiled mode disabled:** While we upgrade to the next OIIO version we need to disable tiled support for the deep driver since it is crashing on OIIO 1.2.2. Note that tiled, deep EXR files are not supported by Nuke anyway. (#4013)
- **Parameters of pointer type not written to .ass:** Since any pointer to memory cannot be preserved by writing and later reading from an .ass file, we have disabled .ass writing for such pointer-type parameters. (#3860)
- **Removed enable\_threaded\_displacement option:** There have been no reported issues requiring this fallback option to be turned off, so we are retiring it and making threaded displacement always on. (#4054)
- **Anisotropic Cook-Torrance BRDF:** Now that the Cook-Torrance BRDF functions actually make use of the anisotropic shading

parameters, they are more sensitive to "junk" data which could cause erroneous results or even crashes if the shader code is not careful. It is recommended that shaders making use of the Cook-Torrance BRDFs be revised so that the normal, tangent and bitangent parameters form an orthonormal basis with each other (when provided), and that both of the roughness parameters take on reasonable values. (#4042)

- **Removed specular\_brdf from standard shader:** Since the Cook-Torrance is similar in performance to the Ward-Duer BRDF yet superior in quality due to its reduced edge darkening and higher energy conservation, the specular\_brdf parameter has been removed from standard, leaving Cook-Torrance as the default (and only!) BRDF of the standard shader. Therefore, the look of existing standard shader-based materials that use the anisotropic Ward-Duer BRDF might change a little bit for the better. (#4042)
- **list bucket scanning mode:** Due to the new bucket threading code, when using the list bucket scanning mode, in-place modification of the bucket list during rendering is no longer possible. This was an unexpected usage of the API. In a future release, we will provide a method to dynamically select buckets during rendering. In the meantime, the list bucket scanning mode is assumed to be given a static list. (#3873)

## Bug fixes

Ticket	Summary
#3866	crash with multiple texture-mapped quad area lights
#3884	Crash with polymesh index arrays bigger than 16M elements
#3950	Crash when interrupting rendering during BVH build
#3987	AiHairDirectDiffuseCache() non-deterministic
#3205	User data inheritance on procedural networks is not working
#3237	Fully transparent samples not properly accounted for in raw深深 drivers
#3268	trace_sets not working for procedurals
#3294	bump2d does not work with Pref when UV coords are not defined
#3370	broken motion_vector AOV in objects with deform keys and ignore_motion_blur
#3766	Deep EXR driver is missing a ZBack channel for Nuke compatibility
#3787	Setting sss_bssrdf_samples to 0 should disable SSS computations
#3814	Crash with invalid camera matrix
#3825	Missing referenced disp shader causes mesh to disappear
#3829	Deep EXR driver uses too much virtual memory
#3830	ignore_motion_blur not removing extra keys
#3835	several second startup delay on certain linux systems even for trivial renders
#3840	curved motion blur is sometimes using a slightly off rotation
#3841	linking quad_light.color to a shader not working with multiple lights
#3843	inaccurate "rays/pixel" progress report during rendering
#3850	unnormalized lights with zero radius should be treated as normalized
#3854	Deep EXR crash when defining data tolerances but no data channels
#3856	maketx crash (windows)
#3863	Light filters incorrectly modify surface UVs
#3864	Volume RGB output fading as step size decreases
#3865	Crash in AiEnd() in free mode after destroying geometry nodes
#3875	Fixed render checkpointing for rare case with multiple AOVs
#3879	Correct memory accounting for sample stores
#3883	falloff radius for constant decay lights should be infinite
#3885	Deep EXR driver with high tolerance has bogus extra sample per pixel
#3888	remove correlation artifacts between pixels
#3896	ignore_motion_blur not enabled at reference_time 0
#3904	Kick display driver crash on Windows with render region
#3905	Faster render abort
#3909	AiThreadSelf() not working properly on Windows
#3915	subdivision numerical precision regression at high iterations
#3916	reported displacement time is larger than it should be
#3919	Deep EXR and raw drivers missing samples from user-defined AOVs
#3923	empty sss_setname not being ignored
#3924	Inconsistent mesh_light intensity when mesh is scaled

#3929	RGB/RGBA user data is gamma corrected when writing to .ass
#3930	Missing AtRGBA operators in python bindings
#3931	Crash writing to .ass with a string param of max allowed length
#3942	allow composition of RGB AOVs connected to RGBA drivers and viceversa
#3946	reported unaccounted memory is too low
#3958	Out-of-range pixel coords in sample store access
#3965	Lights in a procedural with no geometry are not transformed
#3966	Lights in a 2-level procedural network not correctly transformed
#3968	Kick should skip progressive renders when there is no display window
#3972	AiOcclusion artifacts with regular and reflection occlusion
#3975	NaN samples with zero value transmittance in the standard shader
#3989	AiMsg* was ignoring messages before AiBegin
#3997	Interrupting rendering during subdivision causes extraneous warnings and garbled mem stats
#4008	Vector displacement incorrect on UV seams
#4032	Interruption not terminating quickly for subdiv and displacement
#4057	out-of-range error in spotlight when using duplicate keys
#4061	Add pid to logs
#4070	Arnold doesn't accept floats with dot as the last character in IES and metadata files
#4084	Deep AOV sample memory reporting was wrong
#4089	Add missing operators in Python bindings
#4091	Wrong procedural user data with procedural cache enabled
#4093	Light occlusion should behave consistently
#4094	Add parameter type to API interpolation templates
#4101	Deep data tolerances incorrectly set when alpha is also a data channel
#4105	A instance referencing a non-shape node causes a crash
#3743	Report cycles on instances and avoid crash
#3940	Typo in python bindings
#3991	Emit warning when using an array value on a non-array parameter
#4014	Better message when disp_padding is too small
#4087	AiSSSTraceSingleScatter darkening for albedo 1
#4109	rays/pixel stat should be rounded to nearest integer

## 4.2.1.2

### Milestone 4.2.1

September 17, 2014

#### Bug fixes

Ticket	Summary
#4263	black dots/nans near center of spotlight with cosine_power > 0
#4265	8 and 16 bit gamma corrected textures generate some values above 1
#4267	rare random crash at start of rendering

## 4.2.1.1

### Milestone 4.2.1

September 12, 2014

#### Bug fixes

Ticket	Summary
#4260	Overscan output invalid for non-tiled EXRs

## 4.2.1.0

### Milestone 4.2.1

September 8, 2014

#### Enhancements

- **EXR overscan support:** Overscan is now supported by extending the render region beyond the regular image coordinates. E.g. a ten-pixel overscan for a 640x480 image in all directions can be achieved with options `region_min_x,region_min_y,region_max_x,region_max_y` of `-10,-10,649,489`. The old render region defaults of `-1,-1,-1,-1` will continue to render the whole image, but new defaults (`INT_MIN`) are available going forward indicating the render region is not used. EXR output will fully respect the overscan region with differing data and display windows, but the other image formats will simply enlarge the final image resolution to incorporate overscan since they cannot accomodate data outside the display area of the image. (#3102)
- **Improved Cook-Torrance specular and refraction sampling:** Many scenes will now render with significantly reduced noise for glossy reflection and rough refraction with the microfacet-based Cook-Torrance BSDF. In particular, BSDFs viewed at grazing angles or involved in multiple glossy or refraction bounces are improved. (#4064, #4196, #4209)
- **Improved sampling of textured quad lights:** Quad lights now use a new sampling strategy that adaptively chooses the best sample distribution for each shading point. The noise reduction is most noticeable with textured lights, both on surfaces and in volumes. (#3567)
- **Automatic texture\_max\_open\_files:** The options.texture\_max\_open\_files is now set by default to 0, which means that the maximum number of texture files that can be simultaneously opened is automatically computed by Arnold using a new heuristic. This now works for Windows, Linux, and OS X. We expect that the majority of users will be able to leave this at 0 and get the best performance. If Arnold runs out of file handles, Arnold is now more likely to be able to recover without crashing. If the automatic texture\_max\_open\_files that is set is deemed too low, then texture\_max\_open\_files can still be set manually. However, a better solution (and often the only solution) on Linux and OS X is to increase, in the OS, the open file handle hard limit. (#4139, #4140, #4141)
- **Improved performance for shadow ray texture lookups:** For shades done to shadow rays (transparency), texture lookups now use less texture I/O. (#4167)
- **Improved performance for rough glossy texture lookups:** This improvement was supposed to have been in 4.0.12, but due to a bug, it was never actually enabled in the `AiBRDFIntegrate()` function used by the standard shader. Shaders that were manually calling the older `APIAiCookTorranceIntegrate()` were never affected by this bug and have been able to use this improvement since 4.0.12. Now both `AiBRDFIntegrate()` and `AiCookTorranceIntegrate()` make use of this feature. Now that the standard shader finally has this feature, users should find that glossy reflections will pull the correct texture mipmap level across all distances while maintaining an optimal amount of texture sharpness. This can result in dramatic reductions in texture I/O and lessens the need for using the `texture_glossy_blur` option, which we now recommend setting to zero. Unfortunately, third party shaders that do their own integration and manual ray tracing (such as `alSurface` and `Kettle`) will not be helped by these fixes, something that we hope to address in a future version. (#4164, #4176)
- **Faster IPR when rendering with many textures:** Time between IPR progressive frames has been decreased when there are many textures being used. This is especially evident when the textures reside on a high latency file server. (#4152)
- **Low-level optimizations:** Several optimizations were made to improve performance. Some of these are most noticeable on linux and/or when rendering empty or low-complexity regions. (#4193, #4212, #4228, #4149, #1231)
- **Upgraded OIIO to 1.4:** This new version comes with many little improvements and bug fixes. For instance, processing a 54k-resolution TIFF file with `maketx` on Linux is now 4.5x faster compared to the previous version. However, we now require a manual texture flush in order for changes made to texture files be reflected inside of arnold. (#3593, #3646, #3826, #4154)

#### API additions

- **Python 3 support:** The Arnold Python API and pykick are now fully compatible with Python 3, workarounds for module importing and unicode strings are no longer needed. (#4239)

#### Incompatible changes

- **Difference in microfacet BTDFs:** Rough refraction viewed at grazing angles or using Phong-smoothed normals renders slightly darker now, especially for high roughness values. This affects the standard shader and `AiMicrofacetBTDFIntegrate()`, and fixes a bug in the previous BTDF sampling. (#4196, #4231)
- **Handling of unassociated alpha for TGA and PSD:** OpenImageIO 1.4 has more conformant handling of unassociated ("unpremultiplied") alpha in TGA and PSD files. This results in improved texture filtering for typical files, but may also cause TGA files that are missing the appropriate metadata to render darker.

## Bug fixes

Ticket	Summary
#3998	Writing an .ass with open procedurals will not preserve overrides
#4158	Indirect from volumes in SSS is extremely noisy
#3183	OIIO cache refuses to close texture files
#3355	Overridden opaque attribute on procedural not written out
#3605	extreme anisotropy in Cook-Torrance BRDF produces black
#3624	Calling AiUniverseCacheFlush(AI_CACHE_TEXTURE) outside AiBegin / AiEnd crashes
#3674	OIIO locking texture files between renders on windows
#3823	deepexpr driver crashes in tiled mode
#3867	OIIO does not check validity of image SHA
#3893	light_gamma is not applied to skydome_light color
#4068	utility shader unconditionally overwrites opacity when opaque flag is off
#4119	faceting in Cook-Torrance and Ward-Duer speculators
#4123	NaNs in bump mapping when Ns and N differ very slightly
#4126	make backtrace handler more resilient to errors
#4130	zero-length or NaN tangents causing artifacts in Cook-Torrance and Ward-Duer MIS functions
#4131	Zero roughness crashes AiCookTorranceIntegrate
#4134	crash with non-existent procedural path and AI_RENDER_MODE_FREE
#4137	null tangent vectors potentially causing crashes in Cook-Torrance and Ward-Duer
#4142	Don't crash if OIIO runs out of file handles
#4144	Crash after cloning shader with linked array parameter
#4147	Support user data for mesh light texture
#4159	AiShaderGlobalsGet() hang if called during displacement
#4160	Volume node should adjust rays to local space
#4161	Wrong AtPoint2 constructor in Python bindings
#4163	robustness fixes in OBJ procedural loader
#4164	excessive texture I/O in cook-torrance blurry reflections in standard shader
#4171	Sampling position offset should be object space in density shader
#4181	Interactive creation of an object in an empty scene does not work
#4183	Spherical quad sampling converges a different result than area sampling in quad_light
#4184	Enable tiled Deep EXR output
#4186	driver quantization API functions overflow with large float values
#4187	skydome_light crash in IPR and corruption with per-component linked color
#4190	Wrong shadow when using utility shader to modulate opacity of an object
#4194	crash when reporting error before the options node was initialized
#4198	Sky image texture returns NaN when looking at poles
#4205	Degenerate linear or curved motion transforms can cause crashes
#4211	Removing a node with unsolved references causes a crash
#4213	bump2d shifts with highly scaled UV coordinates
#4214	AiNodeGetArray/Matrix returns wrong matrices on instances
#4215	Deep EXR driver crash when reused in the same session
#4222	Shadow precision artifacts with lights with visible geometries
#4231	Microfacet refraction has sharp total internal reflection
#4240	SSS invalid or crashes on non-polymesh objects
#4244	Camera exposure affecting AOV alpha channels
#4251	Skydome interactive enable/disable not working
#4252	Don't write 'options' node default values to .ass file
#4157	Need to warn when passing empty filename as texture name
#4192	Cannot set metadata for "node" parameter on an .mtd file

## 4.2.0.6

### Milestone 4.2

July 16, 2014

#### Enhancements

- **Stereo/multicam rendering:** The output statements have been extended to accept an optional camera name as the first parameter in an outputs string. Custom exporters or host application plugins can use this to setup multi-camera renders. The new additional syntax is:

```
"camera AOV_name AOV_type filter driver" # for a regular driver  
"camera driver" # for a raw driver
```

Multi-camera renders of the same scene can share the reading in and processing of files, acceleration structures, tessellation, displacement and texture cache by having the different cameras render sequentially. Interleaved bucket rendering for different cameras or sending the results from multiple cameras to the same multi-layer driver is not yet supported. (#2404)

- **Improved motion blur sampling quality:** Motion blur sampling has been improved, resulting in less noise given the same AA sample and time budget. The improvement is most prominent in scenes where motion blur is the primary source of noise in the image. There are now two possible values for theAA\_motionblur\_pattern option: the new default pattern, and the legacy jittered pattern which is provided as a backup until the new sampler is sufficiently battle-tested in production. (#2131, #4037, #4044, #4099)
- **Improved glossy sampling quality:** Some scenes will now render with slightly less noise given the same amount of samples and time. In particular, microfacet-based BRDFs viewed at grazing angles will show significant improvement. (#3571, #3888, #3900)
- **Improved CPU utilization:** The thread scheduler has been redesigned achieving much better CPU utilization in scenes that get stuck in a particularly slow bucket near the end of the frame. It is no longer necessary to use very small buckets to prevent these slowdowns. (#3873)
- **Improved memory performance and usage in Linux:** Linux-specific optimizations have been added which reduce memory usage and improve the threading scalability in certain situations, usually during scene preprocessing and shutdown. These require matched AiMalloc() / AiFree() allocations. If you allocate memory in your shader/procedural with AiMalloc() and then free it with free(), it is possible Arnold could crash. (#3779)
- **Faster ray accel build:** Acceleration structure construction is now up to 1.3x faster while using the same memory. (#3898)
- **Faster subdivision:** Catmull-Clark mesh subdivision is now up to 1.5x faster and uses about 5% less peak memory. (#3903, #3913, #3949)
- **Faster displacement:** Polymesh vertex displacement has been sped up, particularly for lower thread counts. On some highly subdivided models we've seen around 2x faster displacement even with 8 threads. In addition, vector displacement artifacts on UV seams have been eliminated. (#4008, #4017)
- **Faster polymesh processing:** Polymesh sanity checks are now about 3x faster, so we have removed the seldomly used options.mesh\_sanity setting and now always perform these checks. Mesh processing/optimization time is now reported in the stats and this step is about 1.5x faster than before. (#3912, #3917, #3927, #3952, #3986)
- **Faster rendering of many-light scenes:** Scenes with many spaced out lights will now render faster thanks to improved culling of low contribution lights. We've seen up to 2x faster renders in some challenging production scenes. (#4100)
- **Faster SSS:** Raytraced diffusion-based SSS should now show higher performance on objects that have not been assigned to be a part of an SSS set via a "sss\_setname" user data string, specially on scenes with spatially overlapping objects, like hair over skin. (#3808)
- **Faster image shader:** Texture map lookups coming from the image shader are now significantly faster. The noise shader has been made slightly faster too. (#3963, #4114)
- **Faster AiNodeGet\* and motion vectors:** AiNodeGet\*() is now substantially faster on multi-threaded machines. Before, excessively calling these functions would cause Arnold to not scale to multiple threads. Now Arnold is able to scale when these functions are called during rendering. There is still a substantial overhead for calling these functions and so AiNodeGet\*() usage is still discouraged from being called during shader evaluate. Usage of AiWorldToScreenMatrix(), AiShaderGlobalsGetPixelMotionVector(), and AiDriverGetMatrices() are all substantially faster now and will scale to multiple threads. One benefit of this is that computing motion vectors is now substantially faster. (#3096, #3836, #3849)
- **Faster string processing:** We now scale better when handling strings. For instance, if many threads are frequently making calls that pass in strings, such as AiNodeDeclare(mynode, "some\_string", "uniform RGB") we will now scale to all these threads and execute this call more quickly. This matters mostly in deferred loading of geometry from DSO/.ass procedural nodes. (#3921, #3957)
- **Faster AiNode():** Performance scaling has been improved when many threads are rapidly calling AiNode(), as it can happen with deferred loading of geometry from DSO/.ass procedural nodes. Under normal situations, this bottleneck is not hit and so no speedup will be seen. (#3954)
- **Faster deep output using less memory:** Renders that output deep images are now up to 15% faster and use less than half the memory. (#4081, #4102)
- **Faster AOV composition with no memory overhead:** The memory overhead incurred when usingoptions.enable\_aov\_composition has been eliminated. This can save gigabytes in highly-threaded, high AA, multi-AOV

renders. Complex renders with lots of transparency and AOVs could be up to 20% faster. (#4038, #4065)

- **Faster image output:** Taking advantage of the non-locking process\_bucket API call introduced in 4.1, we have increased concurrency and improved the speed at which AOVs are written to disk, with a 15% to 30% speedup in built-in drivers like driver\_exr, driver\_tiff, etc. Custom drivers written by third-party developers should be updated accordingly for maximum performance. (#3857)
- **Faster backlighting:** The translucency, or "backlighting" computation in the standard shader, controlled with the Kb parameter, will use on average 50% fewer shadow rays. (#3890)
- **Skip skydome importance table if color is not linked:** Skydome lights will no longer build their importance tables when the color is not linked to anything. This can save a few seconds of preprocessing time when using high resolution tables. Also, IPR renders will now automatically respond to changing the skydome color. (#3020)
- **autobump in SSS:** Arnold can now optionally take into account the effect that autobump has on the ray traced BSSRDF's result via the sss\_use\_autobump\_rend option. This helps capture the high frequency details of the surface more accurately, at the expense of slightly longer render times. For backwards compatibility and performance reasons, this option defaults to false. (#3872, #3894)
- **Smoothen bump near shadow terminator:** We have fixed long-standing faceting issues near the shadow terminator boundary when using strong bump mapping. The improvement is most noticeable with detailed, high-frequency maps or when seen from far away. (#3891)
- **Reduced edge darkening in microfacet BSDFs:** Thanks to a new less aggressive shadowing/masking term, the built-in Cook-Torrance BRDF and microfacet BTDF now conserve a bit more energy and exhibit less edge darkening, particularly at high roughness settings. The impact is expected to be very subtle in most scenes. (#4043)
- **Anisotropic Cook-Torrance BRDF:** The Cook-Torrance BRDF functions in the shading API as well as the standard shader now actually make use of the anisotropic shading parameters. This BRDF has an anisotropic shading effect that is very similar to the anisotropy in the Ward-Duer BRDF in behavior, but with a greater amount of energy conservation and less edge darkening. (#4042)
- **Improved ambient and reflection occlusion:** Ambient occlusion artifacts around edges of low-poly objects have been cleaned up. In addition, the object visibility attributes (visibility, self\_shadows) or the minimum occlusion distance (mint > 0 in the AiOcclusion call itself) can now be used to remove artifacts near silhouettes for reflection occlusion, or to disable occlusion from bump-mapped normals. (#3972)
- **Ambocc distance in utility shader:** Added a new ao\_distance parameter in the utility shader to control the length of ambocc rays for scenes where the default of 100 units is too short. (#3861)
- **Refraction opacity AOV:** The standard shader now can output refraction opacity as an AOV. It is advised to write the complete RGBA set of channels, since the alpha component will constitute a mask that can be used to mix it in with the regular opacity, multiplying in refraction opacity to regular opacity based on the mask. (#2880)
- **Motion-blurred rolling shutter:** Cameras now have a rolling\_shutter\_duration parameter with which it is possible to control the duration of exposure of the scanlines in a rolling shutter camera. Valid values for this parameter are in the 0 to 1 range, where a value of 0 gives you an instantaneous exposure of each scanline (the default value and the rolling shutter's previous behavior), and a value of 1 exposes every scanline for the entirety of the camera's shutter interval (the same result that a camera without rolling shutter would give). (#4009)
- **Support for custom shutter shapes:** Added an additional curve value to the shutter\_type enum in cameras. This, in conjunction with a new camera parameter shutter\_curve, allows arbitrary shutter shapes, linearly interpolating shutter open/closed values between points. You can define as many points as needed. Coordinates are increasing in X from 0 (corresponding to shutter\_start) to 1 (corresponding to shutter\_end), and values in Y must be non-negative. (#3934)

```
# example shutter_curve usage
persp_camera
{
    ...
    # double-trapezoid shutter with outer edges being more gentle,
    # and the first trapezoid emphasized much more
    shutter_type curve
    shutter_curve 8 1 POINT2
        0.0 0.0
        0.2 3.0
        0.3 3.0
        0.4 0.0
        0.6 0.0
        0.7 1.0
        0.8 1.0
        1.0 0.0
}
```

- **Volume plugins:** A new node volume has been added which allows volume plugin DSOs (specified via the dso parameter of a volume node). Volume plugins allow a plugin writer to wrap any volume format or generator for efficient rendering. Volume plugins provide volume creation, destruction, channel sampling, and gathering of ray extents, as well as an automatic bounding box and step size for ray marching (although these can be overridden by the user in the volume node). Ray extent computations inside the plugin will tightly bound volume data along the ray so that the number of volume shader evaluations are drastically reduced in sparse volume datasets. For a short tutorial on how to implement your own volume plugin DSO, see this [article](#) on the arnoldpedia. (#3466, #3978, #3996, #4045, #4046)
- **Density shader:** A new built-in shader density is available which uses AiVolumeSampleChannel() to sample from the new volume plugins. This allows volume sampling to work regardless of what source the volume data comes in (as long as there is a volume

plugin for the source). Currently it has parameters for controlling RGB scattering, absorption, emission, forward vs backward scattering strength, interpolation quality, and offsetting the sampling position (for example with a noise shader to create the appearance of a higher resolution volume than what came from the source data). (#3466, #3994)

- **Static OIIO linking in Windows:** Just like we do in Linux and OSX, we now statically link the OpenImageIO library in Windows too; we don't distribute the file OpenImageIO.dll anymore. (#3969)
- **Scene update on "free" mode:** When using the "free" render mode, you can now force a scene update using AiRender(AI\_RENDER\_MODE\_FREE), which will take into account parameter changes and nodes created or destroyed. (#3862)
- **Multithreading in procedurals:** The enable\_threaded\_procedurals option no longer affects the loading of the text-based .ass, .obj and .plyformats, which are now always loaded in parallel. This option will still apply to binary procedurals (.so/.dll/.dylib), forcing sequential processing when disabled. (#4096)
- **IES files in texture\_searchpath:** If not found in the default path, photometric IES files are now searched for in texture\_searchpath too. (#4036)

## API additions

- **is\_perspective metadata:** View-dependent geometry effects like min\_pixel\_width and adaptive subdivision are now supported in custom camera plugins that have a float fov parameter. This requires the developer of the custom camera to set the boolean is\_perspective metadata to true. An example is given below. NOTE: we reserve the right to supersede this with a better API at some point in the future. (#2756)

```
node_parameters
{
    AiParameterFlt("fov", 60.0);
    AiMetaDataSetBool(mds, NULL, "is_perspective", true);
}
```

- **AiPoint() and AiVector():** A new API function AiPoint() has been added which is aimed at replacing the old and deprecated API AiV3Create(). An equivalent function AiVector() has been added, simply for naming consistency. Analogously, 2D functions AiPoint2() and AiVector2() have also been added. These new APIs allow writing shorter, more concise code in certain cases. (#3936)
- **Volume plugin API:** Volume plugins provide callbacks very similar to procedural plugins. The callbacks are for initialization and takedown of the plugin, initialization and destruction of volumes, sampling of volumes, and providing tight ray extents along a ray where there is volumetric data to sample (viaAiVolumeAddIntersection()). Volume plugins may query declared user parameters from the volume node to customize their loading of volume data as needed. Volume plugins are expected to provide a bounding box around the volume data, as well as a recommended step size for ray marching the data. Shaders may query the volume plugin for data on specific named channels via AiVolumeSampleChannel(). Please see the API documentation for details and this short tutorial [article](#). (#3466)
- **External memory tracking:** A new API function AI\_API void AiAddMemUsage(AtInt64 size, const char\* category) is available for plugin writers and API users to inform Arnold when they allocate and deallocate memory. The string category used will show up in log files under the memory stats at the end of each AiRender() call. It is strongly advised that all shaders, plugins and procedurals that need to allocate memory make use of this new API so that log files can tell the true story of where memory is being used, otherwise it all goes into the "unaccounted" line of the memory stats. (#3948)
- **Procedural with variable number of nodes:** Procedural nodes are now allowed to return 0 from the NumNodes() callback, to avoid having to return a fixed number of nodes when that might not be known in advance. In this case, the GetNode() callback will be called in sequence until it returns NULL. (#3971)
- **Reset parameter to default value:** A new function AiNodeResetParameter() is provided to reset a node parameter to its default value, also removing any links to that parameter. (#4071)

## Incompatible changes

- **Empty sss\_setname strings ignored:** Objects with SSS that have been assigned to be part of an SSS set via an empty "sss\_setname" user data string will no longer be considered as part of the same set, which may result in differences in the rendered result. (#3923)
- **Removed cubic projection map:** This old projection map is rarely used, has seam artifacts along cube edges, and is the least efficient of the various possible cubic formats, so we have decided to remove its support in the sky and skydome\_light nodes. Note that we have left the C API intact,AiMappingCubicMap() and AiMappingCubicMapDerivs(), in order not to break binary compatibility in the rare event that anybody was using these API calls. These calls are now marked as deprecated and will be removed in a future release. (#3937)
- **edgelength changes:** The edge\_length color mode in the utility shader now works on any mesh, regardless of whether it's subdivided or not. This visualization is handy for detecting over-tessellated objects. The old behaviour has been moved to a new mode called pixelerror, which is similar but based on how well the polygon matches subdiv\_pixel\_error. (#3956)
- **Early abort when no outputs are present:** When no output drivers are present the render will now be canceled. This has the side effect that progressivekick renders will be aborted when no display driver is present. If for debugging reasons you wish a render to continue even without valid outputs you can useoptions.abort\_on\_error false. (#3668)
- **Deep EXR driver tiled mode disabled:** While we upgrade to the next OIIO version we need to disable tiled support for the deep driver since it is crashing on OIIO 1.2.2. Note that tiled, deep EXR files are not supported by Nuke anyway. (#4013)

- **Parameters of pointer type not written to .ass:** Since any pointer to memory cannot be preserved by writing and later reading from an .ass file, we have disabled .ass writing for such pointer-type parameters. (#3860)
- **Removed enable\_threaded\_displacement option:** There have been no reported issues requiring this fallback option to be turned off, so we are retiring it and making threaded displacement always on. (#4054)
- **Anisotropic Cook-Torrance BRDF:** Now that the Cook-Torrance BRDF functions actually make use of the anisotropic shading parameters, they are more sensitive to "junk" data which could cause erroneous results or even crashes if the shader code is not careful. It is recommended that shaders making use of the Cook-Torrance BRDFs be revised so that the normal, tangent and bitangent parameters form an orthonormal basis with each other (when provided), and that both of the roughness parameters take on reasonable values. (#4042)
- **Removed specular\_brd from standard shader:** Since the Cook-Torrance is similar in performance to the Ward-Duer BRDF yet superior in quality due to its reduced edge darkening and higher energy conservation, the specular\_brd parameter has been removed from standard, leaving Cook-Torrance as the default (and only!) BRDF of the standard shader. Therefore, the look of existing standard shader-based materials that use the anisotropic Ward-Duer BRDF might change a little bit for the better. (#4042)
- **list bucket scanning mode:** Due to the new bucket threading code, when using the list bucket scanning mode, in-place modification of the bucket list during rendering is no longer possible. This was an unexpected usage of the API. In a future release, we will provide a method to dynamically select buckets during rendering. In the meantime, the list bucket scanning mode is assumed to be given a static list. (#3873)

## Bug fixes

Ticket	Summary
#3866	crash with multiple texture-mapped quad area lights
#3884	Crash with polymesh index arrays bigger than 16M elements
#3950	Crash when interrupting rendering during BVH build
#3987	AiHairDirectDiffuseCache() non-deterministic
#3205	User data inheritance on procedural networks is not working
#3237	Fully transparent samples not properly accounted for in raw深深 drivers
#3268	trace_sets not working for procedurals
#3294	bump2d does not work with Pref when UV coords are not defined
#3370	broken motion_vector AOV in objects with deform keys and ignore_motion_blur
#3766	Deep EXR driver is missing a ZBack channel for Nuke compatibility
#3787	Setting sss_bssrdf_samples to 0 should disable SSS computations
#3814	Crash with invalid camera matrix
#3825	Missing referenced disp shader causes mesh to disappear
#3829	Deep EXR driver uses too much virtual memory
#3830	ignore_motion_blur not removing extra keys
#3835	several second startup delay on certain linux systems even for trivial renders
#3840	curved motion blur is sometimes using a slightly off rotation
#3841	linking quad_light.color to a shader not working with multiple lights
#3843	inaccurate "rays/pixel" progress report during rendering
#3850	unnormalized lights with zero radius should be treated as normalized
#3854	Deep EXR crash when defining data tolerances but no data channels
#3856	maketx crash (windows)
#3863	Light filters incorrectly modify surface UVs
#3864	Volume RGB output fading as step size decreases
#3865	Crash in AiEnd() in free mode after destroying geometry nodes
#3875	Fixed render checkpointing for rare case with multiple AOVs
#3879	Correct memory accounting for sample stores
#3883	falloff radius for constant decay lights should be infinite
#3885	Deep EXR driver with high tolerance has bogus extra sample per pixel
#3888	remove correlation artifacts between pixels
#3896	ignore_motion_blur not enabled at reference_time 0
#3904	Kick display driver crash on Windows with render region
#3905	Faster render abort
#3909	AiThreadSelf() not working properly on Windows

#3915	subdivision numerical precision regression at high iterations
#3916	reported displacement time is larger than it should be
#3919	Deep EXR and raw drivers missing samples from user-defined AOVs
#3923	empty sss_setname not being ignored
#3924	Inconsistent mesh_light intensity when mesh is scaled
#3929	RGB/RGBA user data is gamma corrected when writing to .ass
#3930	Missing AtRGBA operators in python bindings
#3931	Crash writing to .ass with a string param of max allowed length
#3942	allow composition of RGB AOVs connected to RGBA drivers and viceversa
#3946	reported unaccounted memory is too low
#3958	Out-of-range pixel coords in sample store access
#3965	Lights in a procedural with no geometry are not transformed
#3966	Lights in a 2-level procedural network not correctly transformed
#3968	Kick should skip progressive renders when there is no display window
#3972	AiOcclusion artifacts with regular and reflection occlusion
#3975	NaN samples with zero value transmittance in the standard shader
#3989	AiMsg* was ignoring messages before AiBegin
#3997	Interrupting rendering during subdivision causes extraneous warnings and garbled mem stats
#4008	Vector displacement incorrect on UV seams
#4032	Interruption not terminating quickly for subdiv and displacement
#4057	out-of-range error in spotlight when using duplicate keys
#4061	Add pid to logs
#4070	Arnold doesn't accept floats with dot as the last character in IES and metadata files
#4084	Deep AOV sample memory reporting was wrong
#4089	Add missing operators in Python bindings
#4091	Wrong procedural user data with procedural cache enabled
#4093	Light occlusion should behave consistently
#4094	Add parameter type to API interpolation templates
#4101	Deep data tolerances incorrectly set when alpha is also a data channel
#4105	A instance referencing a non-shape node causes a crash
#3743	Report cycles on instances and avoid crash
#3940	Typo in python bindings
#3991	Emit warning when using an array value on a non-array parameter
#4014	Better message when disp_padding is too small
#4087	AiSSSTraceSingleScatter darkening for albedo 1
#4109	rays/pixel stat should be rounded to nearest integer

Ticket Summary	
#4119	faceting in Cook-Torrance and Ward-Duer speculars
#4123	NaNs in bump mapping when Ns and N differ very slightly

Ticket Summary	
#4126	make backtrace handler more resilient to errors
#4130	zero-length or NaN tangents causing artifacts in Cook-Torrance and Ward-Duer MIS functions
#4131	Zero roughness crashes AiCookTorranceIntegrate
#4134	crash with non-existent procedural path and AI_RENDER_MODE_FREE

**Ticket Summary**

#4137 null tangent vectors potentially causing crashes in Cook-Torrance and Ward-Duer

**Ticket Summary**

#4144 Crash after cloning shader with linked array parameter

**Ticket Summary**

#4158 Indirect from volumes in SSS is extremely noisy

#4147 Support user data for mesh light texture

#4159 AiShaderGlobalsGet\*() hang if called during displacement

#4161 Wrong AtPoint2 constructor in Python bindings

**Ticket Summary**

#4160 Volume node should adjust rays to local space

#4171 Sampling position offset should be object space in density shader

#4181 Interactive creation of an object in an empty scene does not work

#4186 driver quantization API functions overflow with large float values

#4187 skydome\_light crash in IPR and corruption with per-component linked color

#4190 Wrong shadow when using utility shader to modulate opacity of an object

#4194 crash when reporting error before the options node was initialized

#4192 Cannot set metadata for "node" parameter on an .mtd file

## 4.1.3.5

### Milestone 4.1.3

April 10, 2014

#### *Enhancements*

- **Faster AiPerlin4:** The 4-dimensional Perlin noise function AiPerlin4() is now about 1.5x faster. (#3777)
- **Faster bump3d:** The bump3d shader now makes 25% fewer shader evaluations, which can reduce render times specially when using expensive procedural maps. (#3804)
- **Faster lights:** Non-motion blurred lights are now a few percent faster to render, specially in scenes with many lights. (#3791, #3792)
- **Reduced per-object memory overhead:** Ray accel structs now consume a few hundred KBs less memory, which can add up to hundreds of MBs saved in scenes with thousands of objects. We also lowered peak memory used in certain situations. (#3772, #3778, #3783)
- **overlay\_mode in utility shader:** We have added a new overlay\_mode parameter to the utility shader which allows you do overlay wireframe on top of the regular color and shading modes. This can be set in the kick command-line with the -om command. (#3800)

#### *Incompatible changes*

- **Behavior of standard.specular\_rotation changed:** The specular\_rotation parameter in the standard shader will now continuously rotate in the same direction when provided values outside of the [0,1] range instead of switching direction. (#3786)
- **objwire utility mode removed:** With the new overlay\_mode, the objwire color mode in the utility shader is now redundant. The same result can be achieved with -cm obj -om polywire. (#3800)
- **Hardcoded name of options node:** The name attribute of the options node is now hardcoded to the string "options" and cannot be redefined. Calls toAiNodeSetStr(AiUniverseGetOptions(),"name","foo") will not have any effect. (#3817)
- **Removed shader\_timing\_stats:** The global option shader\_timing\_stats which was used to report fine-grained timings of certain shading operations has been removed. The accompanying diagnostics AOV "texturetime" has also been removed. This both removes a bit of overhead and simplifies the internal logic. (#3818)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3781	peak memory not properly captured	arnold	thiago	major	4.1	4 months
#3782	corrupted render region output in the X11 kick driver	kick	ramon	major	4.1	4 months
#3784	AiShaderGlobalsGetTriangle crashes with displacement in free mode	arnold	ramon	major	4.1	4 months
#3785	AiRadiance texture coordinates	arnold	ramon	major	4.1	4 months
#3786	specular_rotation parameter in standard shader misbehaving outside of [0,1] range	arnold	alan	major	4.1	4 months
#3788	NaNs with light_blocker	arnold	ramon	major	4.1	4 months
#3789	crash in bump3d used in object space	arnold	ramon	major	4.1	4 months
#3798	static non-invertible matrix transform crashes	arnold	thiago	major	4.1	4 months
#3803	curved motion blur not numerically precise	arnold	thiago	major	4.1	4 months
#3806	fast_exp symbol is stripped from os x opt build	arnold	thiago	major	4.1	4 months
#3809	matrix-transformed cylinder and disk lights broken with 1 motion key	arnold	alan	major	4.1	4 months
#3810	Memory leak in AiNodeEntryInstall	arnold	angel	major	4.1	4 months
#3811	AiNoise reports warning when using distortion	arnold	thiago	major	4.1	4 months
#3812	Don't compute importance tables for disabled lights	arnold	mike	major	4.1	4 months
#3816	Crash when interrupting render during BVH build with more than 2 motion keys	arnold	mike	major	4.1	4 months
#3817	hardcode options node's name to "options"	arnold	marcos	major	4.1	4 months
#3799	Memory leak on NURBS node	arnold	angel	minor	4.1	4 months

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3370	broken motion_vector AOV in objects with deform keys and ignore_motion_blur	4.1.3.1	arnold	thiago	major	4.1.4
#3825	Missing referenced disp shader causes mesh to disappear	4.1.3.1	arnold	mike	major	4.1.4
#3830	ignore_motion_blur not removing extra keys	4.1.3.1	arnold	thiago	major	4.1.4

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3840	curved motion blur is sometimes using a slightly off rotation	4.1.3.2	arnold	thiago	major	4.1.4
#3841	linking quad_light.color to a shader not working with multiple lights	4.1.3.2	arnold	angel	major	4.1.4

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3866	crash with multiple texture-mapped quad area lights	4.1.3.3	arnold	angel	critical	4.1.4
#3766	Deep EXR driver is missing a ZBack channel for Nuke compatibility	MPC DWA 4.1.3.3	arnold	ramon	major	4.1.4
#3854	Deep EXR crash when defining data tolerances but no data channels	4.1.3.3	arnold	ramon	major	4.1.4
#3863	Light filters incorrectly modify surface UVs	4.1.3.3	arnold	ramon	major	4.1.4

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3987	AiHairDirectDiffuseCache() non-deterministic	framestore 4.1.3.4	arnold	alan	critical	4.1.4

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#4057	out-of-range error in spotlight when using duplicate keys	4.1.3.5	arnold	thiago	major	4.1.4

## 4.1.3.3

### Milestone 4.1.3

January 13, 2014

#### *Enhancements*

- **Faster AiPerlin4:** The 4-dimensional Perlin noise function AiPerlin4() is now about 1.5x faster. (#3777)
- **Faster bump3d:** The bump3d shader now makes 25% fewer shader evaluations, which can reduce render times specially when using expensive procedural maps. (#3804)
- **Faster lights:** Non-motion blurred lights are now a few percent faster to render, specially in scenes with many lights. (#3791, #3792)
- **Reduced per-object memory overhead:** Ray accel structs now consume a few hundred KBs less memory, which can add up to hundreds of MBs saved in scenes with thousands of objects. We also lowered peak memory used in certain situations. (#3772, #3778, #3783)
- **overlay\_mode in utility shader:** We have added a new overlay\_mode parameter to the utility shader which allows you do overlay wireframe on top of the regular color and shading modes. This can be set in the kick command-line with the -om command. (#3800)

#### *Incompatible changes*

- **Behavior of standard.specular\_rotation changed:** The specular\_rotation parameter in the standard shader will now continuously rotate in the same direction when provided values outside of the [0,1] range instead of switching direction. (#3786)
- **objwire utility mode removed:** With the new overlay\_mode, the objwire color mode in the utility shader is now redundant. The same result can be achieved with -cm obj -om polywire. (#3800)
- **Hardcoded name of options node:** The name attribute of the options node is now hardcoded to the string "options" and cannot be redefined. Calls toAiNodeSetStr(AiUniverseGetOptions(),"name","foo") will not have any effect. (#3817)
- **Removed shader\_timing\_stats:** The global option shader\_timing\_stats which was used to report fine-grained timings of certain shading operations has been removed. The accompanying diagnostics AOV "texturetime" has also been removed. This both removes a bit of overhead and simplifies the internal logic. (#3818)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3781	peak memory not properly captured	arnold	thiago	major	4.1	7 weeks
#3782	corrupted render region output in the X11 kick driver	kick	ramon	major	4.1	7 weeks
#3784	AiShaderGlobalsGetTriangle crashes with displacement in free mode	arnold	ramon	major	4.1	7 weeks
#3785	AiRadiance texture coordinates	arnold	ramon	major	4.1	7 weeks
#3786	specular_rotation parameter in standard shader misbehaving outside of [0,1] range	arnold	alan	major	4.1	7 weeks
#3788	NaNs with light_blocker	arnold	ramon	major	4.1	7 weeks
#3789	crash in bump3d used in object space	arnold	ramon	major	4.1	7 weeks
#3798	static non-invertible matrix transform crashes	arnold	thiago	major	4.1	6 weeks
#3803	curved motion blur not numerically precise	arnold	thiago	major	4.1	6 weeks
#3806	fast_exp symbol is stripped from os x opt build	arnold	thiago	major	4.1	5 weeks
#3809	matrix-transformed cylinder and disk lights broken with 1 motion key	arnold	alan	major	4.1	5 weeks
#3810	Memory leak in AiNodeEntryInstall	arnold	angel	major	4.1	5 weeks
#3811	AiNoise reports warning when using distortion	arnold	thiago	major	4.1	5 weeks
#3812	Don't compute importance tables for disabled lights	arnold	mike	major	4.1	5 weeks
#3816	Crash when interrupting render during BVH build with more than 2 motion keys	arnold	mike	major	4.1	4 weeks
#3817	hardcode options node's name to "options"	arnold	marcos	major	4.1	4 weeks
#3799	Memory leak on NURBS node	arnold	angel	minor	4.1	6 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3370	broken motion_vector AOV in objects with deform keys and ignore_motion_blur	4.1.3.1	arnold	thiago	major	4.1.4
#3825	Missing referenced disp shader causes mesh to disappear	4.1.3.1	arnold	mike	major	4.1.4
#3830	ignore_motion_blur not removing extra keys	4.1.3.1	arnold	thiago	major	4.1.4

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3840	curved motion blur is sometimes using a slightly off rotation	4.1.3.2	arnold	thiago	major	4.1.4
#3841	linking quad_light.color to a shader not working with multiple lights	4.1.3.2	arnold	angel	major	4.1.4

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3866	crash with multiple texture-mapped quad area lights	4.1.3.3	arnold	angel	critical	4.1.4
#3766	Add ZBack channel to deep EXR images for Nuke compatibility	MPC DWA 4.1.3.3	arnold	ramon	major	4.1.4
#3854	Deep EXR crash when defining data tolerances but no data channels	4.1.3.3	arnold	ramon	major	4.1.4
#3863	Light filters incorrectly modify surface UVs	4.1.3.3	arnold	ramon	major	4.1.4

## 4.1.3.2

### Milestone 4.1.3

December 13, 2013

#### Enhancements

- **Faster AiPerlin4:** The 4-dimensional Perlin noise function AiPerlin4() is now about 1.5x faster. (#3777)
- **Faster bump3d:** The bump3d shader now makes 25% fewer shader evaluations, which can reduce render times specially when using expensive procedural maps. (#3804)
- **Faster lights:** Non-motion blurred lights are now a few percent faster to render, specially in scenes with many lights. (#3791, #3792)
- **Reduced per-object memory overhead:** Ray accel structs now consume a few hundred KBs less memory, which can add up to hundreds of MBs saved in scenes with thousands of objects. We also lowered peak memory used in certain situations. (#3772, #3778, #3783)
- **overlay\_mode in utility shader:** We have added a new overlay\_mode parameter to the utility shader which allows you do overlay wireframe on top of the regular color and shading modes. This can be set in the kick command-line with the -om command. (#3800)

#### Incompatible changes

- **Behavior of standard.specular\_rotation changed:** The specular\_rotation parameter in the standard shader will now continuously rotate in the same direction when provided values outside of the [0,1] range instead of switching direction. (#3786)
- **objwire utility mode removed:** With the new overlay\_mode, the objwire color mode in the utility shader is now redundant. The same result can be achieved with -cm obj -om polywire. (#3800)
- **Hardcoded name of options node:** The name attribute of the options node is now hardcoded to the string "options" and cannot be redefined. Calls toAiNodeSetStr(AiUniverseGetOptions(),"name","foo") will not have any effect. (#3817)
- **Removed shader\_timing\_stats:** The global option shader\_timing\_stats which was used to report fine-grained timings of certain shading operations has been removed. The accompanying diagnostics AOV "texturetime" has also been removed. This both removes a bit of overhead and simplifies the internal logic. (#3818)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3781	peak memory not properly captured	arnold	thiago	major	4.1	5 weeks
#3782	corrupted render region output in the X11 kick driver	kick	ramon	major	4.1	5 weeks
#3784	AiShaderGlobalsGetTriangle crashes with displacement in free mode	arnold	ramon	major	4.1	5 weeks
#3785	AiRadiance texture coordinates	arnold	ramon	major	4.1	5 weeks
#3786	specular_rotation parameter in standard shader misbehaving outside of [0,1] range	arnold	alan	major	4.1	5 weeks
#3788	NaNs with light_blocker	arnold	ramon	major	4.1	5 weeks
#3789	crash in bump3d used in object space	arnold	ramon	major	4.1	5 weeks
#3798	static non-invertible matrix transform crashes	arnold	thiago	major	4.1	4 weeks
#3803	curved motion blur not numerically precise	arnold	thiago	major	4.1	4 weeks
#3806	fast_exp symbol is stripped from os x opt build	arnold	thiago	major	4.1	3 weeks
#3809	matrix-transformed cylinder and disk lights broken with 1 motion key	arnold	alan	major	4.1	3 weeks
#3810	Memory leak in AiNodeEntryInstall	arnold	angel	major	4.1	3 weeks
#3811	AiNoise reports warning when using distortion	arnold	thiago	major	4.1	3 weeks
#3812	Don't compute importance tables for disabled lights	arnold	mike	major	4.1	3 weeks
#3816	Crash when interrupting render during BVH build with more than 2 motion keys	arnold	mike	major	4.1	2 weeks
#3817	hardcode options node's name to "options"	arnold	marcos	major	4.1	2 weeks
#3799	Memory leak on NURBS node	arnold	angel	minor	4.1	4 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3370	broken motion_vector AOV in objects with deform keys and ignore_motion_blur	4.1.3.1	arnold	thiago		

## 4.1.3.1

### Milestone 4.1.3

December 19, 2013

#### Enhancements

- **Faster AiPerlin4:** The 4-dimensional Perlin noise function AiPerlin4() is now about 1.5x faster. (#3777)
- **Faster bump3d:** The bump3d shader now makes 25% fewer shader evaluations, which can reduce render times specially when using expensive procedural maps. (#3804)
- **Faster lights:** Non-motion blurred lights are now a few percent faster to render, specially in scenes with many lights. (#3791, #3792)
- **Reduced per-object memory overhead:** Ray accel structs now consume a few hundred KBs less memory, which can add up to hundreds of MBs saved in scenes with thousands of objects. We also lowered peak memory used in certain situations. (#3772, #3778, #3783)
- **overlay\_mode in utility shader:** We have added a new overlay\_mode parameter to the utility shader which allows you do overlay wireframe on top of the regular color and shading modes. This can be set in the kick command-line with the -om command. (#3800)

#### Incompatible changes

- **Behavior of standard.specular\_rotation changed:** The specular\_rotation parameter in the standard shader will now continuously rotate in the same direction when provided values outside of the [0,1] range instead of switching direction. (#3786)
- **objwire utility mode removed:** With the new overlay\_mode, the objwire color mode in the utility shader is now redundant. The same result can be achieved with -cm obj -om polywire. (#3800)
- **Hardcoded name of options node:** The name attribute of the options node is now hardcoded to the string "options" and cannot be redefined. Calls toAiNodeSetStr(AiUniverseGetOptions(),"name","foo") will not have any effect. (#3817)
- **Removed shader\_timing\_stats:** The global option shader\_timing\_stats which was used to report fine-grained timings of certain shading operations has been removed. The accompanying diagnostics AOV "texturetime" has also been removed. This both removes a bit of overhead and simplifies the internal logic. (#3818)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3781	peak memory not properly captured	arnold	thiago	major	4.1	3 weeks
#3782	corrupted render region output in the X11 kick driver	kick	ramon	major	4.1	3 weeks
#3784	AiShaderGlobalsGetTriangle crashes with displacement in free mode	arnold	ramon	major	4.1	3 weeks
#3785	AiRadiance texture coordinates	arnold	ramon	major	4.1	3 weeks
#3786	specular_rotation parameter in standard shader misbehaving outside of [0,1] range	arnold	alan	major	4.1	3 weeks
#3788	NaNs with light_blocker	arnold	ramon	major	4.1	3 weeks
#3789	crash in bump3d used in object space	arnold	ramon	major	4.1	3 weeks
#3798	static non-invertible matrix transform crashes	arnold	thiago	major	4.1	2 weeks
#3803	curved motion blur not numerically precise	arnold	thiago	major	4.1	2 weeks
#3806	fast_exp symbol is stripped from os x opt build	arnold	thiago	major	4.1	12 days
#3809	matrix-transformed cylinder and disk lights broken with 1 motion key	arnold	alan	major	4.1	10 days
#3810	Memory leak in AiNodeEntryInstall	arnold	angel	major	4.1	9 days
#3811	AiNoise reports warning when using distortion	arnold	thiago	major	4.1	9 days
#3812	Don't compute importance tables for disabled lights	arnold	mike	major	4.1	8 days
#3816	Crash when interrupting render during BVH build with more than 2 motion keys	arnold	mike	major	4.1	6 days
#3817	hardcode options node's name to "options"	arnold	marcos	major	4.1	5 days
#3799	Memory leak on NURBS node	arnold	angel	minor	4.1	2 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3370	broken motion_vector AOV in objects with deform keys and ignore_motion_blur	4.1.3.1	arnold	thiago	major	4.1.4
#3825	Missing referenced disp shader causes mesh to disappear	4.1.3.1	arnold	mike	major	4.1.4
#3830	ignore_motion_blur not removing extra keys	4.1.3.1	arnold	thiago	major	4.1.4



## 4.1.3.0

### Milestone 4.1.3

December 16, 2013

#### *Enhancements*

- **Faster AiPerlin4:** The 4-dimensional Perlin noise function AiPerlin4() is now about 1.5x faster. (#3777)
- **Faster bump3d:** The bump3d shader now makes 25% fewer shader evaluations, which can reduce render times specially when using expensive procedural maps. (#3804)
- **Faster lights:** Non-motion blurred lights are now a few percent faster to render, specially in scenes with many lights. (#3791, #3792)
- **Reduced per-object memory overhead:** Ray accel structs now consume a few hundred KBs less memory, which can add up to hundreds of MBs saved in scenes with thousands of objects. We also lowered peak memory used in certain situations. (#3772, #3778, #3783)
- **overlay\_mode in utility shader:** We have added a new overlay\_mode parameter to the utility shader which allows you do overlay wireframe on top of the regular color and shading modes. This can be set in the kick command-line with the -om command. (#3800)

#### *Incompatible changes*

- **Behavior of standard.specular\_rotation changed:** The specular\_rotation parameter in the standard shader will now continuously rotate in the same direction when provided values outside of the [0,1] range instead of switching direction. (#3786)
- **objwire utility mode removed:** With the new overlay\_mode, the objwire color mode in the utility shader is now redundant. The same result can be achieved with -cm obj -om polywire. (#3800)
- **Hardcoded name of options node:** The name attribute of the options node is now hardcoded to the string "options" and cannot be redefined. Calls toAiNodeSetStr(AiUniverseGetOptions(),"name","foo") will not have any effect. (#3817)
- **Removed shader\_timing\_stats:** The global option shader\_timing\_stats which was used to report fine-grained timings of certain shading operations has been removed. The accompanying diagnostics AOV "texturetime" has also been removed. This both removes a bit of overhead and simplifies the internal logic. (#3818)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3781	peak memory not properly captured	arnold	thiago	major	4.1	3 weeks
#3782	corrupted render region output in the X11 kick driver	kick	ramon	major	4.1	3 weeks
#3784	AiShaderGlobalsGetTriangle crashes with displacement in free mode	arnold	ramon	major	4.1	3 weeks
#3785	AiRadiance texture coordinates	arnold	ramon	major	4.1	3 weeks
#3786	specular_rotation parameter in standard shader misbehaving outside of [0,1] range	arnold	alan	major	4.1	3 weeks
#3788	NaNs with light_blocker	arnold	ramon	major	4.1	3 weeks
#3789	crash in bump3d used in object space	arnold	ramon	major	4.1	3 weeks
#3798	static non-invertible matrix transform crashes	arnold	thiago	major	4.1	13 days
#3803	curved motion blur not numerically precise	arnold	thiago	major	4.1	11 days
#3806	fast_exp symbol is stripped from os x opt build	arnold	thiago	major	4.1	9 days
#3809	matrix-transformed cylinder and disk lights broken with 1 motion key	arnold	alan	major	4.1	7 days
#3810	Memory leak in AiNodeEntryInstall	arnold	angel	major	4.1	6 days
#3811	AiNoise reports warning when using distortion	arnold	thiago	major	4.1	6 days
#3812	Don't compute importance tables for disabled lights	arnold	mike	major	4.1	5 days
#3816	Crash when interrupting render during BVH build with more than 2 motion keys	arnold	mike	major	4.1	3 days
#3817	hardcode options node's name to "options"	arnold	marcos	major	4.1	40 hours
#3799	Memory leak on NURBS node	arnold	angel	minor	4.1	12 days

## 4.1.2.0

### Milestone 4.1.2

November 25, 2013

#### *Enhancements*

- **Windows support for deep EXR2 driver:** The driver\_deepexpr node, which was previously only working for Linux and OSX, now works in Windows too. (#3751)
- **Faster Perlin noise:** AiPerlin3() has been optimized and is now 3.5x faster on top of the 1.4x speedup that was added in 4.1.1, for an accumulated 5x faster over 4.1.0. (#3746)
- **Reduced per-object memory footprint:** We have reduced the per-object memory overhead for instances, polymeshes, curves and procedurals, which will help in scenes with millions of objects. As an added benefit, the ray accel build scalability has been slightly improved on machines with many CPU sockets. (#3770, #3728, #3772)
- **Miscellaneous optimizations in lights:** The light source code has been optimized across the board, including a 7% speedup for scenes with manycylinder\_light nodes, and an important reduction in memory for crazy scenes with hundreds of thousands of lights. (#1261, #3771, #3773)

#### *Bug fixes*

Ticket	Summary	Component	Owner	Priority	Version	Created
#3589	crash in polymesh/ginstance if matrix has null scaling and rotation	arnold	thiago	critical	4.0	3 months
#3258	Interruption leaves some nodes unrenderable during a begin/end session	arnold	mike	major	4.0	10 months
#3768	Deep Driver crash when only Z and A are output	arnold	ramon	major	4.1	4 days
#2772	disallow setting empty strings as a node name	arnold	angel	minor	3.3	20 months

## 4.1.1.0

### Milestone 4.1.1

November 14, 2013

#### Enhancements

- **Reduced polymesh memory footprint:** We now have slightly better compression of polygon mesh data. Up to 1.1x smaller mesh sizes are commonly seen in our tests, although in theory the memory savings could be even larger. We have also saved 64 bytes per polymesh node, 72 bytes per curves node and 56 bytes per all other geometric objects, which helps when rendering very complex scenes with millions of instances, polymeshes or procedurals. (#3675, #3707, #3713, #3715, #3724, #3726, #3733)
- **Faster binary .ass reading:** We can now read b85-compressed integers in large meshes slightly faster. Assuming the .ass file is already in the OS file cache, we've seen up to a 1.3x speedup for .ass files with large meshes. (#3710)
- **Faster Perlin noise:** AiPerlin2() and AiPerlin3() noise is now about 1.3-1.5x faster. (#3746)
- **Faster gobo shader:** The overhead of the gobo shader used for slidemap projection in spotlights has been reduced. This was done at the cost of disallowing shader links in all parameters other than slidemap. (#3712)
- **Blending of RGBA AOVs:** RGBA AOVs now support opacity blending in shaders that have properly declared them to be composable. Note that this will only allow the RGB components of the RGBA value to accumulate through the various semi-opaque layers, since there is currently no way for shaders to define the opacity of the A component of their RGBA result. (#3007)
- **Automatic simplification of motion keys:** Remove motion blur from cameras and lights which have identical motion keys. This is done for consistency with the processing already applied to geometry. This was causing lots of information to be calculated per sample instead of per light. (#3721)
- **Automatic estimation of custom camera ray derivatives:** Developers writing custom camera nodes can leave the dDdx, dDdy, dOdx, and dOdy fields in AtCameraOutput to zero, and an accurate estimate will be computed for them automatically. This prevents extreme texture IO degradation and makes it possible to implement custom cameras without having to worry about the math for computing ray origin and direction derivatives. (#3652)
- **Custom EXR metadata:** An array of strings custom\_attributes has been added to the EXR driver which lets users write their own metadata (#2153). Supported types are int, float, point2, matrix16 and string. Each string has the format "type\_name attr\_name value[s]". Some examples:

```
driver_exr
{
    filename output.exr
    custom_attributes 3 1 STRING
    "float mycustomfloat -23.23"
    "point2 mycustompoint2 -23 -23"
    # will create an EXR string attribute containing "this is my string"
    "string mycustomstring      this is my string"
    ...
}
```

- **Raw drivers now can accept input from user defined AOVs:** When declaring output AOVs in options.outputs the syntax for regular drivers is now also allowed for raw drivers. This means users can add new AOVs at runtime for a raw driver to work with. If needed, this information can be queried through the usual AiOutputIterator\* calls. (#3750)
- **OpenEXR2 deep output:** A new driver driver\_deepexr has been added to output to deep images. It can be declared in options.outputs like a regular driver. Volumetric samples are not yet supported. For now, this driver is only available in Linux and OSX. A brief explanation of the available attributes: (#3756)

BOOL tiled false	Write out tiled or scanline deep images; Nuke only supports scanline deep images
BOOL subpixel_merge true	Nearby subpixel samples will be merged
BOOL use_RGB_opacity false	Write out RGB opacity, rather than just alpha; Nuke can read these images but cannot display them
FLOAT alpha_tolerance 0.01	Alpha tolerance over which samples will not be merged together
FLOAT depth_tolerance 0.01	Depth tolerance over which samples will not be merged together
BOOL alpha_half_precision false	Use 16-bit floats for alpha layer
BOOL depth_half_precision false	Use 16-bit floats for depth layer
FLOAT[] layer_tolerance (empty)	A list of tolerances that will prevent merging for each AOV in options.outputs; if it is a single value it will apply to all layers
BOOL[] layer_enable_filtering (empty)	A list of booleans enabling or disabling filtering for each AOV in options.outputs. Integers, vectors or points are not filtered by default
BOOL[] layer_half_precision (empty)	A list of booleans enabling or disabling 16-bit floats for each AOV in options.outputs. Integers are always full precision

## API additions

- **AtTextureParams channel control:** A new field start\_channel has been added to AtTextureParams which allows a shader writer to choose the starting channel in the texture to sample from, where consecutive channels will be used (1, 3, or 4 depending on single-channel, RGB, or RGBA sampling). This can be useful for e.g. multi-channel textures with more than just RGBA channels. In order to help identify the different channels, a new API function has been added: (#3749)

```
AI_API const char* AiTextureGetChannelName(const char* filename, unsigned int channel_index);
```

## Incompatible changes

- **Changed type of subdiv\_iterations:** The data type of the polymesh parameter subdiv\_iterations has been changed from int (32-bit) to byte (8-bit) to save memory. Applications/plugins that set this parameter with the C or Python API must now call AiNodeSetByte instead of AiNodeSetInt. (#3707)
- **Changed default autobump\_visibility:** The default autobump\_visibility has been changed so that glossy rays by default do not perform autobump. This results in faster renders. Usually this change should not be noticeable because of the blur inherent in glossy reflections, but if it causes unwanted artifacts, it can be reverted by setting the glossy ray bit in autobump\_visibility back to 1 (the integer value would be 223). (#3739)
- **AiNodeEntryGetFilename() can return NULL:** Client code using the AiNodeEntryGetFilename() API function should make sure it checks for theNULL pointer, which is used to indicate a built-in node. (#3715)
- **AtTextureParams new field:** The new field start\_channel in AtTextureParams will be initialized to zero by default when usingAiTextureParamsSetDefaults(). But if your shaders are initializing the fields of that structure manually, you will need to set this new field to zero or else you may get random start channels when sampling textures. The size of the structure is unchanged, so this should be the only side effect of the new field. (#3749)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2727	inconsistent emission in standard shader when bounce_factor=0	arnold	borja	major	3.3	21 months
#2789	crash in gobo shader with empty rotate array	arnold	marcos	major	3.3	20 months
#3006	crash when firing shadow rays when shading shadow rays	arnold	ramon	major	4.0	15 months
#3007	Support RGBA aov blending through auto transparency	arnold	alan	major	4.0	15 months
#3351	Ward-Duer BRDF in standard shader crashes with infinite dPdu or dPdv values	arnold	thiago	major	4.0	8 months
#3574	per-component linking not working within arrays	arnold	thiago	major	4.0	3 months
#3632	crash in procedural with lights but no geometry	arnold	angel	major	4.0	6 weeks
#3694	metadata errors should be warnings	arnold	angel	major	4.0	4 weeks
#3696	Crash when re-rendering a texture-mapped quad_light	arnold	mike	major	4.0	3 weeks
#3701	rotation matrix decomposition crash	arnold	thiago	major	4.0	3 weeks
#3702	Fix instancing stats for nurbs	arnold	mike	major	4.0	3 weeks
#3703	Replacing shader in an object does not affect its instances	arnold	angel	major	4.0	3 weeks
#3705	Avoid vertical aliasing with rolling shutter	arnold	oscar	major	4.0	3 weeks
#3717	rare crash in malformed .ass file with unmatched square bracket	arnold	angel	major	4.0	2 weeks
#3722	allow '(' and ')' characters in user-data parameter names	arnold	marcos	major	4.0	2 weeks
#3723	bump3d does not work correctly with transformed geometry	arnold	ramon	major	4.0	2 weeks
#3729	bump3d can generate erroneous normals pointing below the surface	arnold	ramon	major	4.0	13 days
#3730	don't write 'threads' and 'ignore_list' options in .ass files	arnold	marcos	major	4.0	13 days
#3734	Memory growing too much with many mesh lights	arnold	ramon	major	4.0	11 days
#3740	Shader override broken for multiple levels of instancing	arnold	angel	major	4.0	8 days
#3742	Visibility override broken for multiple levels of instancing	arnold	angel	major	4.0	7 days
#3745	subdivision creates nans in mesh's dPdu/v	arnold	thiago	major	4.0	7 days
#3757	Procedural override of "transform_time_samples" parameter	arnold	angel	major	4.0	2 hours
#3695	kick -repeat concatenates kick command line in logs	arnold	oscar	trivial	4.0	3 weeks
#3700	Amend AiASSWrite() in the Python API	arnold	oscar	trivial	4.0	3 weeks

# 4.1.0.0

## Milestone 4.1

### *Enhancements*

- **Smoother rotational motion blur:** Interpolation between two motion blur keys now better accounts for rotation. This improvement is very obvious when rotating a propeller blade by close to 180 degrees, since now the blade properly follows the curved path. We can only interpolate angles of up to 180 degrees, so to get a 360 degree rotation (or more), will still take multiple keys. However, the number of keys required can now be greatly reduced. This also prevents certain crashes from occurring where the less precise interpolation caused degenerate non-invertible matrices to be produced. This feature can be turned off by setting the options variable `curved_motionblur` to false. If we do not hear of any problems with this new feature (which is enabled by default), we will eventually get rid of the `curved_motionblur` option and make this the only option. So please let us know if you encounter issues with this. (#3604)
- **Rolling shutter:** It is now possible to simulate the type of rolling shutter effect seen in footage shot with digital cameras that use CMOS-based sensors such as Blackmagics, Alexas, REDs, and even iPhones. This is controlled by the new `rolling_shutter` camera enum parameter, which defaults to "off" and can be set to "top" (top-to-bottom being the most common scanning direction), "bottom", "left" or "right". (#3127)
- **Texture support in mesh\_light:** In a similar way to other mappable lights, a mesh light's color attribute can now be linked to a texture map. A single color will be assigned per mesh triangle, so the resolution of the mesh can have an effect on the quality of the illumination. (#3552)
- **Reduced per-thread memory footprint :** Reduced the memory pool overhead. Savings are most noticeable for large AA sampling rates and large bucket sizes. For an 8x8 AA samples, and 64x64 bucket size render, about 5.3MB is saved per thread. For 32 threads, that is 170MB saved. Also improved the accuracy for memory reporting of memory pools. (#3594)
- **Faster multi-layered SSS:** It is now possible to compute a weighted sum of an arbitrary number of SSS profiles in a single function call via importance sampling, without having to fire a new set of samples for each diffusion radius. This allows for a result that is similar in quality to multiple invocations of the classic cubic/gaussian SSS lookups, but at a fraction of the cost. The speedup is linear on the number of layers. (#3640)
- **Faster .ass file load:** Both .ass and .ass.gz files should now load 2x to 3x faster. (#3532, #3570, #3579, #3581)
- **Reduced per-pixel overhead:** We have added a number of optimizations to the processing, storage and filtering of subpixel samples, AOVs, and camera ray creation. The result is a slight speedup for all renders, and a more significant speedup for pixels that have simpler geometry and/or shaders. For example, a mostly empty black frame with no AOVs now renders 2x faster, and the same scene but using 20 AOVs renders 6x faster. (#3585, #3602, #3614, #3618, #3633)
- **Faster light sampling:** The sampling of spot lights and area lights has been optimized. This can result in a 15% speedup in scenes with many lights. (#3617, #3683)
- **Faster physical\_sky:** The `physical_sky` shader has been optimized and is now 30% faster. In practice, this speedup only applies to the precomputation of the importance tables of the `skydome_light`, specially when large importance table resolutions are used (4k and higher) as is often necessary to capture the small sun disk of the physical sky. (#3529)
- **Faster skydome\_light:** A performance regression was introduced in Arnold 4.0.16, where texture mapped skydome lights would unnecessarily trace shadow rays to black parts of the sky. This is now fixed, and performance should be back to pre-4.0.16 levels. (#3555)
- **Faster back diffuse:** The `Kb` (backlighting) section of the standard shader is now more performant in some cases. When used in conjunction with both front diffuse and specular it will cast one-third fewer shadow rays. It also got a minor speed bump by not evaluating shader networks attached to `Kb` more than once. (#3569)
- **Faster adaptive subdivision:** Several small optimizations in the subdivision engine have resulted in an aggregated 10% speedup when using adaptive subdivision. (#3547)
- **Cached per-component shader links:** Redundant shader evaluations that occurred when the same shader is attached to multiple components of another shader's input (e.g. to `Kd_color.r`, `g`, and `b`) have now been eliminated. This can lead to significant speedups in complex shader networks that abused per-component linking, something that has been observed in the workflow of some Maya artists. (#3657)
- **Volume ray differentials:** Gobo textures applied to spotlights now make use of mipmapping and `mipmap_bias`, therefore reducing the amount of texture I/O. (#3067)
- **Raised texture cache to 1GB and texture\_max\_sharpen to 1.5:** The default value for `options.texture_max_memory_MB` was 512 MB which is now likely too small for complex production shots and could cause cache thrashing. We have raised this default to 1 GB. Very complex film projects may want to increase this manually to 2 or even 4 GB. We have also raised the default `max_texture_sharpen` from 1.0 to 1.5. This will produce sharper, nicer looking textures, but it will also result in increased texture usage -- sometimes by around 2x. If the increased texture usage causes performance problems, this can be manually lowered back to 1.0. (#3584)
- **objwire utility shading mode:** This new mode in the utility shader combines polywire and obj shading in one mode. (#3676)
- **uniformid utility shading mode:** This new mode in the utility shader allows to color by patch instead of by polygon, by curve instead of curve segment. This mirrors how uniform user data is looked up. (#3686)
- **Smart multi-platform shared libraries file extensions:** Shared libraries loaded through `AiLoadPlugins()` or through the `dso` parameter in `proceduralnodes` now accept all regular platform file extensions (.dll, .so, .dylib) regardless of the host platform. Custom extensions are also allowed. (#3432)
- **Report C/C++ standard libraries in Linux:** The version/build information in the log file was incorrectly reporting the version of the GNU C standard library (glibc) installed in the system instead of the version with which Arnold was built. In addition, we now report

- the version of the GNU C++ library (libstdc++) with which Arnold was built. (#3553)
- **Remove useless matrix warnings:** To avoid confusion and polluting log files, we no longer issue performance warnings when identity matrices are supplied or when the motion blur matrices are identical. The overhead when this occurs is insignificant. Note that these matrices are silently removed to ensure optimal performance during ray tracing. (#3650)
  - **Upgraded OIIO to 1.2.2:** This OpenImageIO upgrade comes with several bug fixes, more accurate texture filtering, faster texture I/O and up to 10x faster maketx processing for large images. It also no longer exports the OIIO symbols, see Incompatible changes below for more details. (#3514, #3645, #2898, #3655)
  - **Upgraded RLM to 10.1BL2:** We have upgraded the license server and the external library controlling the licensing subsystem from version 9.4BL4 to 10.1BL2, a more stable release fixing various crashes, bugs, hangs and memory leaks. (#3619)

## API additions

- **Return values for AiArraySet\***: The AiArraySet\* API functions now return false when they fail. This lets applications/exporters print out more useful information about the context in which the failure occurred. (#2684)
- **Return value for AiUniverseCacheFlush**: This function now returns an error code, in particular it will return false when called during a render in progress. (#3361)
- **binary argument in AiASSWrite()**: A new binary argument has been added to AiASSWrite() which controls whether arrays use binary encoding when writing an .ass file. The old global option binary\_ass which used to serve this purpose has been removed. Since -set options.binary\_ass off no longer works, we have added a new kick option to disable binary encoding when re-saving .ass files: kick in.ass -db -resave out.ass. (#2828)
- **Texture blur, mipmap mode, and fill**: The AtTextureParams struct now has additional parameters blur\_s, blur\_t, mipmap\_mode, and fill. While the width parameters that were there already will multiply the filter width, the new blur parameters will add to it. The mipmap\_mode is for fine-tuning MIP filtering; in some cases you know you won't need high-quality MIP filtering, and so you can choose a faster, more approximate mode instead. The fill value is used for non-existent channels in a texture, such as the alpha channel in an RGB-only texture. (#3191)
- **Light cache resets**: Shader light loops will generally automatically cache lighting data, or invalidate the cached data for a subsequent light loop if certain common items in the shader globals change (such as trace sets, surface normal, sg->fhem, etc). Occasionally shader writers wish to get fresh lighting data for a light loop where the cache does not automatically reset; for this purpose we have provided a new API call to ensure that lighting data is recomputed. Note that calling this unnecessarily will obviously defeat the cache and impact performance, so it should only be used as strictly necessary. (#3557)

```
AI_API void AiLightsResetCache(AtShaderGlobals* sg);
```

- **Surface normal derivatives**: Added the surface normal derivatives with respect to image space (dNdx and dNdy) to the AtShaderGlobals struct. (#3193)
- **AiShaderGlobalsGetPolygon() with 256+ vertices**: The existing AiShaderGlobalsGetPolygon() API function now works on polygons with more than 255 vertices. If a NULL pointer is passed as vertex array, the function will simply return the number of vertices in the polygon. This way, an array could be allocated for the right number of vertices (using AiShaderGlobalsQuickAlloc() for speed) and passed in a second call to AiShaderGlobalsGetPolygon() to store the actual vertices into the array. (#3468)
- **AiColor()**: To save a few keystrokes, a new API function AiColor() has been added which is exactly the same as the older AiColorCreate() but shorter. AiColorCreate() is now deprecated and will be removed in the next API-breaking release. (#3653)
- **DriverNeedsBucket() and DriverProcessBucket()**: All custom output drivers must implement these two additional methods. DriverNeedsBucket() tells the renderer whether a driver needs a bucket to be rendered (perhaps because the bucket is already available) and DriverProcessBucket() gives the custom driver a chance to preprocess a bucket in a non-locking way. (#3627)
- **AiNodeSetAttributes() in Python**: When this function was introduced in the C API, we forgot to add the corresponding Python binding. (#3586)

## Incompatible changes

- **Changes in .ass files**: The size of binary-encoded .ass files has been reduced slightly. Among other changes, binary-encoded arrays are now written as a single big line of ASCII characters. This new Arnold version will read older .ass files which used the previous multiline format, but please note that the newly written .ass files will not be readable by older Arnold versions, and in fact may crash while parsing. (#3542, #3609)
- **Reversed latlong and angular environment mapping**: The u axis of AiMappingLatLong(), AiMappingLatLongDerivs(), AiMappingAngularMap() and AiMappingAngularMapDerivs() have been flipped so that these projections behave as expected of environment maps. (#3577)
- **Setting light samples to 0 disables the light**: This is more consistent with how every other sampling knob works. Previously the samples were silently clamped to 1. (#3687)
- **Corrected AiNodesLinked() and AiNodeGetLink() behavior**: The behaviour of these functions has been modified to remove ambiguities related to per-component links. When an RGB parameter was linked with AiNodeLink(image, "Kd\_color", lambert), querying the link stored in the individual R, G and B components of the parameter would incorrectly return the node linked in the parameter itself. With the new behaviour, a call to AiNodesLinked(lambert, "Kd\_color.r") will return false, and a call to AiNodeGetLink(lambert, "Kd\_color.r") will return NULL. (#2699)
- **Removed deprecated AiUniverse functions**: The legacy API functions AiUniverseGetLights, AiUniverseGetNumLights, AiUniverseGetNumGObjects, AiGetNumInstalledNodes and AiNodeEntryLookUpByIndex have been removed. Use the AtNodeIterator and AtNodeEntryIterator API instead. (#1940)
- **Removed external library symbols**: On linux and osx, OpenImageIO, TBB, and all other external libraries no longer have their

symbols exported by Arnold. This means that on these platforms there can no longer be any symbol clashes with these libraries, but it also means that code that relied on these symbols being exported by Arnold must now supply their own version of these libraries. Windows still relies on a dynamic library for OIIO, but we have already removed its headers and libs and it is our intention to remove the DLL in a future version as well. (#2898)

- **Removed legacy/unused options:** We have removed a bunch of legacy global options that are no longer useful. These are AA\_sampling\_dither, GI\_hemi\_pattern, GI\_Cranley\_Patterson, enable\_legacy\_vector\_noise, physically\_based, auto\_transparency\_probabilistic, strict\_procedural\_compatibility, enable\_memory\_reporting and ignore\_mis. (#3597)
- **Removed stretched-Phong BRDF:** The stretched\_phong value of the specular\_brdf enum in the standard shader, that has been deprecated for a couple of years, has finally been removed. (#3630)
- **Removed AiUDataSet\*() functions:** These methods were not thread-safe, and were rarely used, so they caused more problems than they solved. However, no mechanism exists now to send extra data between displacement and surface shaders. (#2740)
- **Removed specialized (and slow) memory reporting support for older linux:** On older distros, such as CentOS 5, Arnold will now report 0MB memory use in the per-line memory stat of the log file. We feel this is preferable to the 10% slowdown incurred by the tons of slow system calls. (#3160)
- **Removed ambient\_light:** The legacy ambient\_light node has finally been removed, along with its associated API AiAmbient(). (#3656)
- **Replaced doubles with floats:** Constants, such as AI\_PI, are now floats instead of doubles. Likewise, many functions that had doubles as arguments or returned doubles, now use floats. Most of these changes should be transparent, but AiSamplerGetSample() now uses float\* arguments, so in order for this to compile, this function must be given floats. (#3663, #3666)
- **Removed deprecated types and functions:** The legacy typedefs of basic data types in ai\_types.h (AtVoid, AtFloat, AtDouble, AtBoolean, Atlnt etc) have finally been removed after being deprecated almost two years ago in Arnold 4.0. Any client code using them will therefore break, and a text replacement for the simpler built-in data types (void, float, double, bool, int etc) will be required. The deprecated legacy functions in ai\_types.h are also gone, specifically BILERP(), BIHERP(), MAP01(), MAP01F(), ACOSF(), INVERSE(), AiBerpScalar(). The deprecated macros TRUE and FALSE have also been removed, which should be replaced in client code for the lowercase true and false. (#3670)
- **Removed unused AtRay fields:** The AtRay struct no longer has the lx, ly, xfrm\_len, and inv\_xfrm\_len fields since they appear to never be used. (#3667)
- **Removed RGB AOV in favor of RGBA:** The internal, built-in AOV "RGB" has been removed in favor of always using "RGBA". Requests for the "RGB" AOV will be redirected to "RGBA" AOV, and a warning will be issued advising to use "RGBA". Also, a new parameter skip\_alpha has been added to the driver\_tiff node to help preserve the old behaviour. (#3688)
- **Replaced dither\_amplitude with boolean dither:** Dithering for 8-bit images can now be turned on and off in output drivers using the boolean ditherattribute. Also, the kick option -da has been removed. (#3691)
- **Removed pointcloud SSS:** Support for the legacy pointcloud-based SSS rendering method has been removed. The rendering of BSSRDFs is now only done using the much simpler ray traced method that was added in Arnold 4.0.7. This change removes the following options: sss\_sample\_spacing (object setting), sss\_sample\_distribution (object setting), sss\_faceset (optional object setting), sss\_threaded\_sample\_distribution (global setting), sss\_sample\_factor (global setting), show\_samples (global setting), sss\_subpixel\_cache (global setting). (#3628)
- **Removed AiSSSPointCloudLookupCubic and AiSSSPointCloudLookupGaussian():** These two functions have been replaced by the following more general functions, which return the weighted sum of an arbitrary number of SSS profiles instead of just three: (#3640)

```
AI_API AtColor AiBSSRDFCubic(const AtShaderGlobals* sg, const float* radius, const AtColor* weight, unsigned int num = 1);
AI_API AtColor AiBSSRDFGaussian(const AtShaderGlobals* sg, const float* variance, const AtColor* weight, unsigned int num = 1);
```

As an example of usage, the code snippet below would have to be changed from this:

```
AtColor sss_radius = AiShaderEvalParamRGB(p_sss_radius);
AtColor Ksss = AiShaderEvalParamRGB(p_sss_color);
AtColor sss_weight = sss_weight;
AtColor sss = AiSSSPointCloudLookupCubic(sg, sss_radius) * sss_weight;
```

To this:

```
AtColor sss_radius = AiShaderEvalParamRGB(p_sss_radius);
AtColor Ksss = AiShaderEvalParamRGB(p_sss_color);
AtColor sss_weight[3];
sss_weight[0] = AiColor(Ksss.r, 0, 0);
sss_weight[1] = AiColor(0, Ksss.g, 0);
sss_weight[2] = AiColor(0, 0, Ksss.b);
AtColor sss = AiBSSRDFCubic(sg,
    &sss_radius[0], // <-- array of sample radii, starting with the first
    component (red) of AtColor
    sss_weight, // <-- array of sample weights, one for each radius
    3); // <-- AiSSSLookupCubic() was actually sampling three
radii in one call
```

And here's how the sum of 6 gaussians to simulate diffusion in skin from Chapter 14 of GPU Gems 3 could be implemented:

```

float variance[6] = {0.0064f, 0.0484f, 0.187f, 0.567f, 1.99f, 7.41f};
    AtRGB weight[6] = {{0.233f, 0.455f, 0.649f},
                        {0.100f, 0.336f, 0.344f},
                        {0.118f, 0.198f, 0 },
                        {0.113f, 0.007f, 0.007f},
                        {0.358f, 0.004f, 0 },
                        {0.078f, 0 , 0 }};
AtRGB sss = AiBSSRDFGaussian(sg, variance, weight, 6);

```

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3588	crash when preprocessing polymeshes with 2 motion keys	arnold	thiago	critical	4.0	2 months
#3610	crash at render shutdown in procedurals with user-data of type array	arnold	angel	critical	4.0	5 weeks
#2699	wrong AiNodeGetLink & AiNodesLinked behavior	arnold	angel	major	3.3	20 months
#2740	Remove support for setting user data in displacement shaders	arnold	mike	major	4.0	20 months
#2760	64bit sizes (like memory allocations) should be of type size_t	arnold	thiago	major	3.3	19 months
#3126	Lights inside procedurals are not affected by the procedural matrix	arnold	angel	major	4.0	12 months
#3245	User data is not cloned with AiNodeClone()	arnold	angel	major	4.0	9 months
#3277	OIIO cache flushing not always works	arnold	ramon	major	4.0	8 months
#3361	AiUniverseCacheFlush() crashes while rendering	arnold	ramon	major	4.0	7 months
#3386	Transparent objects close to the ground plane of the fog atmosphere render with artifacts	arnold	alan	major	4.0	6 months
#3391	AiArraySetMtx crashes when array is NULL	arnold	marcos	major	4.0	6 months
#3425	".ass parsing (deferred)" stat was not properly accounted for	arnold	thiago	major	4.0	6 months
#3464	make AiShaderGlobalsGet*() API functions more robust towards changes in shader globals	arnold	alan	major	4.0	5 months
#3468	Fix support for polygons with >255 vertices in the API	arnold	angel	major	4.0	5 months
#3520	Random dither amplitude in output drivers is too strong	arnold	ramon	major	4.0	4 months
#3525	Arnold won't abort after exceeding max warnings/errors	arnold	thiago	major	4.0	3 months
#3534	clipping planes not affecting volumetrics	arnold	alan	major	4.0	3 months
#3537	Using AiNodeSetStr on a string array leads to a crash	arnold	angel	major	4.0	3 months
#3543	Correctly apply gamma in jpeg driver	arnold	ramon	major	4.0	3 months
#3548	BRDF sampling does not respect mesh_light orientation	arnold	ramon	major	4.0	3 months
#3549	IES parser errors cause crashes	arnold	oscar	major	4.0	3 months
#3555	skydome should take into account low_light_threshold	arnold	thiago	major	4.0	3 months
#3561	Oren-Nayar MIS functions do not work outside of a light loop	arnold	alan	major	4.0	3 months
#3562	Crash when AiNodesLinked() is used with an empty string	arnold	angel	major	4.0	3 months
#3577	latlong and angular environment mappings should be flipped in U	arnold	alan	major	4.0	2 months
#3586	Expose AiNodeSetAttributes() in the Python API	arnold	oscar	major	4.0	2 months
#3587	procedural_force_expand is not written out to .ass files correctly	arnold	oscar	major	4.0	2 months
#3590	IPR crash when changing bucket scanning from hilbert to something else	arnold	pal	major	4.0	2 months
#3591	bump2d crashes when used with curves	arnold	oscar	major	4.0	7 weeks
#3592	multi-element shader arrays not inherited from procedurals	arnold	angel	major	4.0	7 weeks
#3608	Crash with massive non-tiled exr driver	arnold	ramon	major	4.0	5 weeks
#3615	Clean up 'long' and 'unsigned long' usage	arnold	thiago	major	4.0	4 weeks
#3626	clip spot_light cone exactly at origin if lens_radius is > 0	arnold	ramon	major	4.0	3 weeks
#3645	OIIO reports memory wrong (windows)	oiio	ramon	major	4.0	2 weeks
#3648	Crash when a user data is declared and not defined	arnold	oscar	major	4.0	2 weeks
#3658	AtColor/AtPoint/AtPoint2 const operator[] could not be used as L-value	arnold	alan	major	4.0	11 days
#3689	EXR driver causes new threads to be spawned	arnold	ramon	major	4.0	37 hours
#3690	indirect multiplier not taken into account in skydome_light	arnold	alan	major	4.0	29 hours
#3511	crash with illegal values of curves.max_subdivs	arnold	ramon	minor	4.0	4 months
#3583	report .ass.gz being loaded when .ass doesn't exist	arnold	oscar	minor	4.0	2 months
#3638	-set does not work with matrix parameters	kick	alan	minor	4.0	2 weeks

## 4.0.16.2

### Milestone 4.0.16

#### Enhancements

- **Optimized geometry storage:** Certain types of meshes now require less memory thanks to improved mesh compression. In some cases, we've seen dramatic reductions in the storage for normals and UVs. In exchange, this will increase the preprocessing time for mesh loading by around 3 seconds per 50 million faces. In addition, the constant per-object memory overhead for geometry nodes has been slightly reduced which can help in complex scenes with millions of instances, polymeshes or procedurals. (#3428, #3460, #1833)
- **Faster area lights:** Area lights more consistently and correctly work with options.low\_light\_threshold. This can result in significantly faster renders when there are many lights. Compared to previous versions, there is a chance that the resulting image might be brighter or dimmer when using a very aggressive low\_light\_threshold. If it is brighter, then the image is actually closer to the true reference image (when low\_light\_threshold is 0), but if it is darker then low\_light\_threshold was too aggressive and will need to be lowered. For renders using less aggressive settings, such as the default .001 threshold, the image should be identical but render times should be improved. (#3471, #3477)
- **Initial support for IES lights:** We have added a new type of light, the photometric\_light node. This light can initially load IESNA LM-63 photometric data through the parameter filename. (#2906, #3524)
- **Per-strand shader assignments on curves shape:** Similar to how polymesh shapes with multiple shaders are defined, the curves shape now has ashidxs parameter with which shaders from the shader array may be assigned to each separate strand. (#3456)
- **Improvements to the physical\_sky shader:** Added a sun\_size parameter to the physical\_sky shader that allows specifying the angular diameter of the sun in degrees. Also added X, Y and Z parameters, similar to those of the sky shader, with which it is possible to define the orientation of the physical\_sky shader, which was previously hardcoded to "Y is up". (#3452)
- **User-data access for volume shaders:** It is now possible to read user data fields from volumetric shapes, allowing things like per-particle user data on volumetric spherical point clouds to affect the result of volumetric shading. *Note: As is the case of surface shaders, accessing user data from a volume shader is still much slower than access to the shader's own parameters, so discretion is advised.* (#3327)
- **Per-face-vertex user data:** A new storage class, *indexed* can be used for declared user data, where it may be different across shared edges of polygons. This allows for such things as alternate UV sets, pre-baked tangent vectors, and so forth. It works the same way as the built-in vlist/vidxs, uvlist/uvidxs, nlist/nidxs arrays. You only need to declare the indexed data, e.g. *declare altuvlist indexed POINT2*, and then both the list and indexes array can then just be used: *altuvlist 4 1 POINT2 ... altuvidxs 16 1 UINT ....* Note that it follows a strict naming convention; the list array can be named what you want, or end with *list*, and the *list* suffix will be removed and *idxs* appended for the actual index array. Examples of pairs would be *altuvlist* with *altuvidxs*, and *altuv* and *altuvidxs*. (#2866)
- **Multiple UV sets in the image node:** With the new indexed user-data storage class, alternate UVs can be created as POINT2 data on polymesh nodes. These alternate UVs can be selected in the image node by setting their name in the new uvset parameter. For example, if you have created a secondary UV set in a polymesh node and named it "UVset2", then you can use this UV set by setting image.uvset equal to "UVset2". By default, when the uvsetparameter is empty, the primary UV set in the polymesh will be used, i.e. the one stored in the built-in polymesh parameters uvlist and uvidxs, so that backwards compatibility is preserved. (#3500)
- **Warn of millions of warnings:** Warning messages, even if hidden by explicitly setting a low maximum number of warnings, still take time to process. If there are millions of warning messages, typically from a custom shader's shader\_evaluate method that is being too verbose, then this can introduce a sizable render time overhead. Even worse, this overhead does not scale well with increasing numbers of threads. When there are more than a million messages, we now print a performance warning with the amount of predicated time overhead the messages are causing. We have also improved the scaling, but not the overhead, when the majority of the messages are of the same type (the most common case we have seen in practice). Still, the most efficient solution is to never print warnings from a shader's shader\_evaluate method. (#3499, #3501)
- **Upgraded OIIO to 0.10.19:** There have been a few bugfixes and performance improvements since the last OIIO version we shipped with Arnold (which was 0.10.16) including:
  - Minor performance improvements
  - Detailed OIIO stats sometimes causing crashes (#3473)
  - maketx: Fix an overflow problem when running on images larger than 32k x 32k (#3506)
- **kick command line in logs and display window:** Log files generated from kick now contain the entire command-line that was used which can help with debugging. In addition, the command-line is also printed in the title bar of the render display window that is created by the kick process. (#3457)

### API additions

- **Added user-data derivatives with respect to screen x and y:** Derivatives of user data with respect to screen coordinates can now be queried. This should allow more accurate filtering based on user data, such as when using alternate UV sets for texture mapping. This is analogous to the texture derivatives of the primary set of UV coordinates that are stored automatically in sg->dudx,dudy,dvdx,dvdy. These derivatives will be zero except for varying (per-vertex) and indexed (per-face-vertex) user data. They also will be zero during displacement, which will effectively reduce any use of them to point sampling, which may negatively affect performance; we plan on addressing this in a future release. The following functions have been added to the shading API to obtain these derivatives, which will return *true* if the derivatives were available: (#3244)

```

bool AiUDataGetDxyDerivativesFlt(const char* name, float* dx_val, float* dy_val);
bool AiUDataGetDxyDerivativesRGB(const char* name, AtRGB* dx_val, AtRGB* dy_val);
bool AiUDataGetDxyDerivativesRGBA(const char* name, AtRGBA* dx_val, AtRGBA* dy_val);
bool AiUDataGetDxyDerivativesVec(const char* name, AtVector* dx_val, AtVector* dy_val);
bool AiUDataGetDxyDerivativesPnt(const char* name, AtPoint* dx_val, AtPoint* dy_val);
bool AiUDataGetDxyDerivativesPnt2(const char* name, AtPoint2* dx_val, AtPoint2* dy_val);
bool AiUDataGetDxyDerivativesArray(const char* name, AtArray** dx_val, AtArray** dy_val);
bool AiUDataGetDxyDerivativesMatrix(const char* name, AtMatrixx* dx_val, AtMatrix* dy_val);

```

## Incompatible changes

- std::vector inclusion removed from ai\_license.h:** This inclusion is no longer needed by ai\_license.h itself and has been removed. This does not affect binary compatibility, but may cause client code that explicitly included ai\_license.h to require explicitly including std::vector if recompiled. (#3488)
- ai\_version.h inclusion removed from ai\_node\_entry.h and ai\_nodes.h:** That inclusion is not really needed in those files. This does not affect binary compatibility, but may cause client code that were not explicitly including ai\_version.h and using its API to require explicitly including it now. (#2082)
- Renamed physical\_sky parameters:** The parameter solar\_direction has been renamed to sun\_direction, and visible\_solar\_disc has been renamed to enable\_sun. (#3452)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3429	mesh intersection can create bad barycentric coordinates	arnold	ramon	major	4.0	5 months
#3433	Cloning a node with a linked parameter causes a crash when destroying nodes	arnold	angel	major	4.0	5 months
#3448	Cannot load Arnold Python module with unset LD_LIBRARY_PATH	arnold	oscar	major	4.0	5 months
#3458	shadow ring artifact with non-black shadow_color	arnold	thiago	major	4.0	4 months
#3472	lights should take into account _normalize when computing falloff radius	arnold	thiago	major	4.0	4 months
#3473	Printing detailed OIIO stats sometimes crashes	oiio	ramon	major	4.0	4 months
#3475	Add pixel aspect ratio to EXR images	arnold	ramon	major	4.0	4 months
#3482	MIS samples identical for transparent surfaces along a ray	arnold	ramon	major	4.0	4 months
#3483	distortion in skydome light's sample mapping causing severe flickering	arnold	alan	major	4.0	4 months
#3486	infinite lights should not be affected by low_light_threshold	arnold	thiago	major	4.0	4 months
#3491	Polymesh Differentials generate NaN / Inf	arnold	ramon	major	4.0	4 months
#3504	atmosphere lit by disk light not rendering against background until geometry is hit	arnold	thiago	major	4.0	3 months
#3506	maketx crashes with big images	oiio	ramon	major	4.0	3 months
#3509	time-based displacement does not work with duplicate deformation keys	arnold	ramon	major	4.0	3 months
#3513	Division by zero in the density-based volume sampler	arnold	alan	major	4.0	3 months
#3515	User data is not preserved by the procedural cache	arnold	angel	major	4.0	3 months
#3518	volume scattering dark or missing when scaled to a large size	arnold	alan	major	4.0	3 months
#3519	Memory leaks when updating lights	arnold	oscar	major	4.0	3 months
#3522	undefine MIN/MAX macros in public API to avoid symbol clashes	arnold	thiago	major	4.0	3 months
#3526	mesh light crashes while in free render mode	arnold	thiago	major	4.0	2 months
#3527	Fix shader message accumulation in free mode	arnold	ramon	major	4.0	2 months
#3508	remove harmless warning when using base85 encoding of packed INT arrays	arnold	angel	minor	4.0	3 months

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3588	crash when preprocessing polymeshes with 2 motion keys	framestore, 4.0.16.1	arnold	thiago	critical	4.0.17

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3610	crash at render shutdown in procedurals with user-data of type array	4.0.16.2	arnold	angel	major	4.0.17

## 4.0.16.1

### Milestone 4.0.16

#### Enhancements

- **Optimized geometry storage:** Certain types of meshes now require less memory thanks to improved mesh compression. In some cases, we've seen dramatic reductions in the storage for normals and UVs. In exchange, this will increase the preprocessing time for mesh loading by around 3 seconds per 50 million faces. In addition, the constant per-object memory overhead for geometry nodes has been slightly reduced which can help in complex scenes with millions of instances, polymeshes or procedurals. (#3428, #3460, #1833)
- **Faster area lights:** Area lights more consistently and correctly work with options.low\_light\_threshold. This can result in significantly faster renders when there are many lights. Compared to previous versions, there is a chance that the resulting image might be brighter or dimmer when using a very aggressive `low_light_threshold`. If it is brighter, then the image is actually closer to the true reference image (when `low_light_threshold` is 0), but if it is darker then `low_light_threshold` was too aggressive and will need to be lowered. For renders using less aggressive settings, such as the default .001 threshold, the image should be identical but render times should be improved. (#3471, #3477)
- **Initial support for IES lights:** We have added a new type of light, the photometric\_light node. This light can initially load IESNA LM-63 photometric data through the parameter filename. (#2906, #3524)
- **Per-strand shader assignments on curves shape:** Similar to how polymesh shapes with multiple shaders are defined, the curves shape now has `ashidxs` parameter with which shaders from the shader array may be assigned to each separate strand. (#3456)
- **Improvements to the physical\_sky shader:** Added a `sun_size` parameter to the `physical_sky` shader that allows specifying the angular diameter of the sun in degrees. Also added X, Y and Z parameters, similar to those of the `sky` shader, with which it is possible to define the orientation of the `physical_sky` shader, which was previously hardcoded to "Y is up". (#3452)
- **User-data access for volume shaders:** It is now possible to read user data fields from volumetric shapes, allowing things like per-particle user data on volumetric spherical point clouds to affect the result of volumetric shading. *Note: As is the case of surface shaders, accessing user data from a volume shader is still much slower than access to the shader's own parameters, so discretion is advised.* (#3327)
- **Per-face-vertex user data:** A new storage class, `indexed` can be used for declared user data, where it may be different across shared edges of polygons. This allows for such things as alternate UV sets, pre-baked tangent vectors, and so forth. It works the same way as the built-in `vlist/vidxs`, `uvlist/uvidxs`, `nlist/nidxs` arrays. You only need to declare the indexed data, e.g. `declare altuvlist indexed POINT2`, and then both the list and indexes array can then just be used: `altuvlist 4 1 POINT2 ... altuvidxs 16 1 UINT ....` Note that it follows a strict naming convention; the list array can be named what you want, or end with `list`, and the `list` suffix will be removed and `idxs` appended for the actual index array. Examples of pairs would be `altuvlist` with `altuvidxs`, and `altuv` and `altuvidxs`. (#2866)
- **Multiple UV sets in the image node:** With the new indexed user-data storage class, alternate UVs can be created as `POINT2` data on polymesh nodes. These alternate UVs can be selected in the image node by setting their name in the new `uvset` parameter. For example, if you have created a secondary UV set in a polymesh node and named it "UVset2", then you can use this UV set by setting `image.uvset` equal to "UVset2". By default, when the `uvset` parameter is empty, the primary UV set in the polymesh will be used, i.e. the one stored in the built-in polymesh parameters `uvlist` and `uvidxs`, so that backwards compatibility is preserved. (#3500)
- **Warn of millions of warnings:** Warning messages, even if hidden by explicitly setting a low maximum number of warnings, still take time to process. If there are millions of warning messages, typically from a custom shader's `shader_evaluate` method that is being too verbose, then this can introduce a sizable render time overhead. Even worse, this overhead does not scale well with increasing numbers of threads. When there are more than a million messages, we now print a performance warning with the amount of predicated time overhead the messages are causing. We have also improved the scaling, but not the overhead, when the majority of the messages are of the same type (the most common case we have seen in practice). Still, the most efficient solution is to never print warnings from a shader's `shader_evaluate` method. (#3499, #3501)
- **Upgraded OIIO to 0.10.19:** There have been a few bugfixes and performance improvements since the last OIIO version we shipped with Arnold (which was 0.10.16) including:
  - Minor performance improvements
  - Detailed OIIO stats sometimes causing crashes (#3473)
  - maketx: Fix an overflow problem when running on images larger than 32k x 32k (#3506)
- **kick command line in logs and display window:** Log files generated from kick now contain the entire command-line that was used which can help with debugging. In addition, the command-line is also printed in the title bar of the render display window that is created by the kick process. (#3457)

#### API additions

- **Added user-data derivatives with respect to screen x and y:** Derivatives of user data with respect to screen coordinates can now be queried. This should allow more accurate filtering based on user data, such as when using alternate UV sets for texture mapping. This is analogous to the texture derivatives of the primary set of UV coordinates that are stored automatically in `sg->dudx,dudy,dvdx,dvdy`. These derivatives will be zero except for varying (per-vertex) and indexed (per-face-vertex) user data.

They also will be zero during displacement, which will effectively reduce any use of them to point sampling, which may negatively affect performance; we plan on addressing this in a future release. The following functions have been added to the shading API to obtain these derivatives, which will return *true* if the derivatives were available: (#3244)

```
bool AiUDataGetDxyDerivativesFlt(const char* name, float* dx_val, float* dy_val);
bool AiUDataGetDxyDerivativesRGB(const char* name, AtRGB* dx_val, AtRGB* dy_val);
bool AiUDataGetDxyDerivativesRGBA(const char* name, AtRGBA* dx_val, AtRGBA* dy_val);
bool AiUDataGetDxyDerivativesVec(const char* name, AtVector* dx_val, AtVector* dy_val);
bool AiUDataGetDxyDerivativesPnt(const char* name, AtPoint* dx_val, AtPoint* dy_val);
bool AiUDataGetDxyDerivativesPnt2(const char* name, AtPoint2* dx_val, AtPoint2* dy_val);
bool AiUDataGetDxyDerivativesArray(const char* name, AtArray** dx_val, AtArray** dy_val);
bool AiUDataGetDxyDerivativesMatrix(const char* name, AtMatrix* dx_val, AtMatrix* dy_val);
```

## Incompatible changes

- std::vector inclusion removed from ai\_license.h:** This inclusion is no longer needed by ai\_license.h itself and has been removed. This does not affect binary compatibility, but may cause client code that explicitly included ai\_license.h to require explicitly including std::vector if recompiled. (#3488)
- ai\_version.h inclusion removed from ai\_node\_entry.h and ai\_nodes.h:** That inclusion is not really needed in those files. This does not affect binary compatibility, but may cause client code that were not explicitly including ai\_version.h and using its API to require explicitly including it now. (#2082)
- Renamed physical\_sky parameters:** The parameter solar\_direction has been renamed to sun\_direction, and visible\_solar\_disc has been renamed to enable\_sun. (#3452)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3429	mesh intersection can create bad barycentric coordinates	arnold	ramon	major	4.0	4 months
#3433	Cloning a node with a linked parameter causes a crash when destroying nodes	arnold	angel	major	4.0	4 months
#3448	Cannot load Arnold Python module with unset LD_LIBRARY_PATH	arnold	oscar	major	4.0	4 months
#3458	shadow ring artifact with non-black shadow_color	arnold	thiago	major	4.0	3 months
#3472	lights should take into account _normalize when computing falloff radius	arnold	thiago	major	4.0	3 months
#3473	Printing detailed OIIO stats sometimes crashes	oiio	ramon	major	4.0	3 months
#3475	Add pixel aspect ratio to EXR images	arnold	ramon	major	4.0	3 months
#3482	MIS samples identical for transparent surfaces along a ray	arnold	ramon	major	4.0	3 months
#3483	distortion in skydome light's sample mapping causing severe flickering	arnold	alan	major	4.0	3 months
#3486	infinite lights should not be affected by low_light_threshold	arnold	thiago	major	4.0	3 months
#3491	Polymesh Differentials generate NaN / Inf	arnold	ramon	major	4.0	3 months
#3504	atmosphere lit by disk light not rendering against background until geometry is hit	arnold	thiago	major	4.0	2 months
#3506	maketx crashes with big images	oiio	ramon	major	4.0	2 months
#3509	time-based displacement does not work with duplicate deformation keys	arnold	ramon	major	4.0	2 months
#3513	Division by zero in the density-based volume sampler	arnold	alan	major	4.0	2 months
#3515	User data is not preserved by the procedural cache	arnold	angel	major	4.0	8 weeks
#3518	volume scattering dark or missing when scaled to a large size	arnold	alan	major	4.0	8 weeks
#3519	Memory leaks when updating lights	arnold	oscar	major	4.0	8 weeks
#3522	undefine MIN/MAX macros in public API to avoid symbol clashes	arnold	thiago	major	4.0	7 weeks
#3526	mesh light crashes while in free render mode	arnold	thiago	major	4.0	7 weeks
#3527	Fix shader message accumulation in free mode	arnold	ramon	major	4.0	6 weeks
#3508	remove harmless warning when using base85 encoding of packed INT arrays	arnold	angel	minor	4.0	2 months

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3588	crash when preprocessing polymeshes with 2 motion keys	framestore, 4.0.16.1	arnold	thiago	critical	4.0.17

# 4.0.16.0

## Milestone 4.0.16

### Enhancements

- **Optimized geometry storage:** Certain types of meshes now require less memory thanks to improved mesh compression. In some cases, we've seen dramatic reductions in the storage for normals and UVs. In exchange, this will increase the preprocessing time for mesh loading by around 3 seconds per 50 million faces. In addition, the constant per-object memory overhead for geometry nodes has been slightly reduced which can help in complex scenes with millions of instances, polymeshes or procedurals. (#3428, #3460, #1833)
- **Faster area lights:** Area lights more consistently and correctly work with options.low\_light\_threshold. This can result in significantly faster renders when there are many lights. Compared to previous versions, there is a chance that the resulting image might be brighter or dimmer when using a very aggressive low\_light\_threshold. If it is brighter, then the image is actually closer to the true reference image (when low\_light\_threshold is 0), but if it is darker then low\_light\_threshold was too aggressive and will need to be lowered. For renders using less aggressive settings, such as the default .001 threshold, the image should be identical but render times should be improved. (#3471, #3477)
- **Initial support for IES lights:** We have added a new type of light, the photometric\_light node. This light can initially load IESNA LM-63 photometric data through the parameter filename. (#2906, #3524)
- **Per-strand shader assignments on curves shape:** Similar to how polymesh shapes with multiple shaders are defined, the curves shape now has a shidxs parameter with which shaders from the shader array may be assigned to each separate strand. (#3456)
- **Improvements to the physical\_sky shader:** Added a sun\_size parameter to the physical\_sky shader that allows specifying the angular diameter of the sun in degrees. Also added X, Y and Z parameters, similar to those of the sky shader, with which it is possible to define the orientation of the physical\_sky shader, which was previously hardcoded to "Y is up". (#3452)
- **User-data access for volume shaders:** It is now possible to read user data fields from volumetric shapes, allowing things like per-particle user data on volumetric spherical point clouds to affect the result of volumetric shading. *Note: As is the case of surface shaders, accessing user data from a volume shader is still much slower than access to the shader's own parameters, so discretion is advised.* (#3327)
- **Per-face-vertex user data:** A new storage class, *indexed* can be used for declared user data, where it may be different across shared edges of polygons. This allows for such things as alternate UV sets, pre-baked tangent vectors, and so forth. It works the same way as the built-in vlist/vidxs, uvlist/uvidxs, nlist/nidxs arrays. You only need to declare the indexed data, e.g. *declare altuvlist indexed POINT2*, and then both the list and indexes array can then just be used: *altuvlist 4 1 POINT2 ... altuvidxs 16 1 UINT ....* Note that it follows a strict naming convention; the list array can be named what you want, or end with *list*, and the *list* suffix will be removed and *idxs* appended for the actual index array. Examples of pairs would be *altuvlist* with *altuvidxs*, and *altuv* and *altuvidxs*. (#2866)
- **Multiple UV sets in the image node:** With the new indexed user-data storage class, alternate UVs can be created as POINT2 data on polymesh nodes. These alternate UVs can be selected in the image node by setting their name in the new uvset parameter. For example, if you have created a secondary UV set in a polymesh node and named it "UVset2", then you can use this UV set by setting image.uvset equal to "UVset2". By default, when the uvset parameter is empty, the primary UV set in the polymesh will be used, i.e. the one stored in the built-in polymesh parametersuvlist and uvidxs, so that backwards compatibility is preserved. (#3500)
- **Warn of millions of warnings:** Warning messages, even if hidden by explicitly setting a low maximum number of warnings, still take time to process. If there are millions of warning messages, typically from a custom shader's shader\_evaluate method that is being too verbose, then this can introduce a sizable render time overhead. Even worse, this overhead does not scale well with increasing numbers of threads. When there are more than a million messages, we now print a performance warning with the amount of predicated time overhead the messages are causing. We have also improved the scaling, but not the overhead, when the majority of the messages are of the same type (the most common case we have seen in practice). Still, the most efficient solution is to never print warnings from a shader's shader\_evaluate method. (#3499, #3501)
- **Upgraded OIIO to 0.10.19:** There have been a few bugfixes and performance improvements since the last OIIO version we shipped with Arnold (which was 0.10.16) including:
  - Minor performance improvements
  - Detailed OIIO stats sometimes causing crashes (#3473)
  - maketx: Fix an overflow problem when running on images larger than 32k x 32k (#3506)
- **kick command line in logs and display window:** Log files generated from kick now contain the entire command-line that was used which can help with debugging. In addition, the command-line is also printed in the title bar of the render display window that is created by the kick process. (#3457)

### API additions

- **Added user-data derivatives with respect to screen x and y:** Derivatives of user data with respect to screen coordinates can now be queried. This should allow more accurate filtering based on user data, such as when using alternate UV sets for texture mapping. This is analogous to the texture derivatives of the primary set of UV coordinates that are stored automatically in sg->dudx,dudy,dvdx,dvdy. These derivatives will be zero except for varying (per-vertex) and indexed (per-face-vertex) user data. They also will be zero during displacement, which will effectively reduce any use of them to point sampling, which may negatively affect performance; we plan on addressing this in a future release. The following functions have been added to the shading API to obtain these derivatives, which will return *true* if the derivatives were available: (#3244)

```

bool AiUDataGetDxyDerivativesFlt(const char* name, float* dx_val, float* dy_val);
bool AiUDataGetDxyDerivativesRGB(const char* name, AtRGB* dx_val, AtRGB* dy_val);
bool AiUDataGetDxyDerivativesRGBA(const char* name, AtRGBA* dx_val, AtRGBA* dy_val);
bool AiUDataGetDxyDerivativesVec(const char* name, AtVector* dx_val, AtVector* dy_val);
bool AiUDataGetDxyDerivativesPnt(const char* name, AtPoint* dx_val, AtPoint* dy_val);
bool AiUDataGetDxyDerivativesPnt2(const char* name, AtPoint2* dx_val, AtPoint2* dy_val);
bool AiUDataGetDxyDerivativesArray(const char* name, AtArray** dx_val, AtArray** dy_val);
bool AiUDataGetDxyDerivativesMatrix(const char* name, AtMatrixx* dx_val, AtMatrix* dy_val);

```

### Incompatible changes

- std::vector inclusion removed from ai\_license.h:** This inclusion is no longer needed by ai\_license.h itself and has been removed. This does not affect binary compatibility, but may cause client code that explicitly included ai\_license.h to require explicitly including std::vector if recompiled. (#3488)
- ai\_version.h inclusion removed from ai\_node\_entry.h and ai\_nodes.h:** That inclusion is not really needed in those files. This does not affect binary compatibility, but may cause client code that were not explicitly including ai\_version.h and using its API to require explicitly including it now. (#2082)
- Renamed physical\_sky parameters:** The parameter solar\_direction has been renamed to sun\_direction, and visible\_solar\_disc has been renamed to enable\_sun. (#3452)

### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3429	mesh intersection can create bad barycentric coordinates	arnold	ramon	major	4.0	2 months
#3433	Cloning a node with a linked parameter causes a crash when destroying nodes	arnold	angel	major	4.0	2 months
#3448	Cannot load Arnold Python module with unset LD_LIBRARY_PATH	arnold	oscar	major	4.0	2 months
#3458	shadow ring artifact with non-black shadow_color	arnold	thiago	major	4.0	8 weeks
#3472	lights should take into account _normalize when computing falloff radius	arnold	thiago	major	4.0	6 weeks
#3473	Printing detailed OIIO stats sometimes crashes	oiio	ramon	major	4.0	6 weeks
#3475	Add pixel aspect ratio to EXR images	arnold	ramon	major	4.0	6 weeks
#3482	MIS samples identical for transparent surfaces along a ray	arnold	ramon	major	4.0	6 weeks
#3483	distortion in skydome light's sample mapping causing severe flickering	arnold	alan	major	4.0	6 weeks
#3486	infinite lights should not be affected by low_light_threshold	arnold	thiago	major	4.0	5 weeks
#3491	Polymesh Differentials generate NaN / Inf	arnold	ramon	major	4.0	5 weeks
#3504	atmosphere lit by disk light not rendering against background until geometry is hit	arnold	thiago	major	4.0	4 weeks
#3506	maketx crashes with big images	oiio	ramon	major	4.0	3 weeks
#3509	time-based displacement does not work with duplicate deformation keys	arnold	ramon	major	4.0	2 weeks
#3513	Division by zero in the density-based volume sampler	arnold	alan	major	4.0	13 days
#3515	User data is not preserved by the procedural cache	arnold	angel	major	4.0	12 days
#3518	volume scattering dark or missing when scaled to a large size	arnold	alan	major	4.0	11 days
#3519	Memory leaks when updating lights	arnold	oscar	major	4.0	8 days
#3522	undefine MIN/MAX macros in public API to avoid symbol clashes	arnold	thiago	major	4.0	7 days
#3526	mesh light crashes while in free render mode	arnold	thiago	major	4.0	24 hours
#3527	Fix shader message accumulation in free mode	arnold	ramon	major	4.0	6 hours
#3508	remove harmless warning when using base85 encoding of packed INT arrays	arnold	angel	minor	4.0	3 weeks

## 4.0.15.1

### Milestone 4.0.15

#### Enhancements

- **Optimized polymesh storage:** There is an important reduction in polymesh memory use that is proportional to the total number of triangles generated. In addition, the memory use for the polymesh ray accel structures has been reduced by up to 2x. (#3408, #3420)
- **Faster ray accel build:** The polymesh ray accel build time has been reduced by up to 2x without hurting ray tracing intersection performance. (#3420)
- **Faster curvy curves:** Individual curve segments (as opposed to entire curves) that are sufficiently curvy should now render faster. (#3414)
- **Faster pixel filtering:** Samples on the edge of buckets which do not contribute to the image are no longer rendered, speeding up rendering with filter sizes which are not an exact odd number. (#3354)
- **Faster solid angle-based sampling:** The math for the solid angle-based sampling of quad area lights has been optimized resulting in slightly faster render times. This essentially removes any overhead of the new sampling technique which was introduced in the previous release. (#3406)
- **Improved volume sampling for quad and disk lights:** The importance sampling distribution used for rendering volumetric quad and disk lights has been improved, reducing the variance in the visibly noisy region near the light itself. (#3393, #3436)
- **New physically-based sky shader:** A new shader called physical\_sky has been added to the core that implements a physically based sun and sky model. Basic controls for this shader are the color of the ground, the atmospheric haze, and the position of the sun in the sky. The sun's position can be configured either through the solar\_direction vector, or a pair of azimuth and elevation coordinates with valid values in the 0-360° and 0-180° range, respectively. The amount of atmospheric haze can be controlled via the turbidity parameter, with valid values in the 1-10 range. physical\_sky is hard-coded to be invisible to GI rays, so to use it as a light source you must attach it to a skydome\_light with sufficient resolution to capture the small solar disk. Note : *This shader is in active development and will likely suffer compatibility-breaking changes in future versions, so please consider it as a technology preview and not a final implementation, although feedback is of course more than welcome.* (#3446)
- **Optimized compiler flags:** We now use more aggressive compiler optimization flags which result in up to 20% faster renders on Linux and 3% faster on Windows. (#3178, #3424)
- **Error on long EXR channel names:** When an AOV is output to an EXR file with a filename longer than 31 characters, the channels could get mixed up leading to a monochrome result. Arnold now errors at the start of a render if this could happen. Note that this limit will be increased to 255 chars once we upgrade to EXR2. (#3445)
- **Improved crash reporting when out of memory:** When running out of memory and crashing, the stacktrace report now provides a clear error message with how much memory we are using and how much is physically available. This should make users realize exactly what's going on. (#3444)

#### API additions

#### Incompatible changes

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3403	Occasional crash with deform_time_samples covering partial shutter range	arnold	mike	major	4.0	8 weeks
#3405	bounding box of plane primitive is wrong when normal has a -1 component	arnold	thiago	major	4.0	8 weeks
#3407	incorrect ambient occlusion shading with exponential falloff	arnold	thiago	major	4.0	7 weeks
#3409	Bindings are not Python 3.x compatible	arnold	oscar	major	4.0	7 weeks
#3410	quantization of images truncates instead of rounds	arnold	thiago	major	4.0	7 weeks
#3412	Cook-Torrance MIS functions reflect more light than they receive at near 0 roughness	arnold	alan	major	4.0	7 weeks
#3413	camera exposure not working with semi-opaque regions above the background	arnold	alan	major	4.0	7 weeks
#3415	secondary specular in hair shader missing from specular AOV	arnold	marcos	major	4.0	7 weeks
#3418	rays tangent to a box should return the front facing normal	arnold	thiago	major	4.0	7 weeks
#3423	Scaling double-counted for oriented curves	arnold	mike	major	4.0	6 weeks
#3437	motion blurred quad_light renders incorrectly with enable_fast_lights	arnold	thiago	major	4.0	5 weeks
#3442	standard material edge darkening	arnold	alan	major	4.0	5 weeks
#3385	non-deterministic ordering of overlapping primitives across multiple renders	arnold	thiago	minor	4.0	2 months
#3449	shadow ray avg. and max hits stat broken	arnold	marcos	minor	4.0	4 weeks
#3443	remove unused shadow_bias option	arnold	marcos	trivial	4.0	5 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3429	mesh intersection can create bad barycentric coordinates	rgh, 4.0.15.1	arnold	ramon	major	4.0.16
#3433	Cloning a node with a linked parameter causes a crash when destroying nodes	4.0.15.1	arnold	angel	major	4.0.16
#3458	shadow ring artifact with non-black shadow_color	4.0.15.1	arnold	thiago	major	4.0.16
#3475	Add pixel aspect ratio to EXR images	oiio, 4.0.15.1	arnold	ramon	major	4.0.16
#3482	MIS samples identical for transparent surfaces along a ray	framestore, 4.0.15.1	arnold	ramon	major	4.0.16
#3483	distortion in skydome light's sample mapping causing severe flickering	4.0.15.1	arnold	alan	major	4.0.16

# 4.0.15.0

## Milestone 4.0.15

### *Enhancements*

- Optimized polymesh storage: There is an important reduction in polymesh memory use that is proportional to the total number of triangles generated. In addition, the memory use for the polymesh ray accel structures has been reduced by up to 2x. (#3408, #3420)
- Faster ray accel build: The polymesh ray accel build time has been reduced by up to 2x without hurting ray tracing intersection performance. (#3420)
- Faster curvy curves: Individual curve segments (as opposed to entire curves) that are sufficiently curvy should now render faster. (#3414)
- Faster pixel filtering: Samples on the edge of buckets which do not contribute to the image are no longer rendered, speeding up rendering with filter sizes which are not an exact odd number. (#3354)
- Faster solid angle-based sampling: The math for the solid angle-based sampling of quad area lights has been optimized resulting in slightly faster render times. This essentially removes any overhead of the new sampling technique which was introduced in the previous release. (#3406)
- Improved volume sampling for quad and disk lights: The importance sampling distribution used for rendering volumetric quad and disk lights has been improved, reducing the variance in the visibly noisy region near the light itself. (#3393, #3436)
- New physically-based sky shader: A new shader called physical\_sky has been added to the core that implements a physically based sun and sky model. Basic controls for this shader are the color of the ground, the atmospheric haze, and the position of the sun in the sky. The sun's position can be configured either through the solar\_direction vector, or a pair of azimuth and elevation coordinates with valid values in the 0-360° and 0-180° range. physical\_sky is hard-coded to be invisible to GI rays, so to use it as a light source you must attach it to a skydome\_light with sufficient resolution to capture the small solar disk. Note: This shader is in active development and will likely suffer compatibility-breaking changes in future versions, so please consider it as a technology preview and not a final implementation, although feedback is of course more than welcome. (#3446)
- Optimized compiler flags: We now use more aggressive compiler optimization flags which result in up to 20% faster renders on Linux and 3% faster on Windows. (#3178, #3424)
- Error on long EXR channel names: When an AOV is output to an EXR file with a filename longer than 31 characters, the channels could get mixed up leading to a monochrome result. Arnold now errors at the start of a render if this could happen. Note that this limit will be increased to 255 chars once we upgrade to EXR2. (#3445)
- Improved crash reporting when out of memory: When running out of memory and crashing, the stacktrace report now provides a clear error message with how much memory we are using and how much is physically available. This should make users realize exactly what's going on. (#3444)

### *API additions*

### *Incompatible changes*

### *Bug fixes*

Ticket	Summary	Component	Owner	Priority	Version	Created
#3403	Occasional crash with deform_time_samples covering partial shutter range	arnold	mike	major	4.0	4 weeks
#3405	bounding box of plane primitive is wrong when normal has a -1 component	arnold	thiago	major	4.0	4 weeks
#3407	incorrect ambient occlusion shading with exponential falloff	arnold	thiago	major	4.0	3 weeks
#3409	Bindings are not Python 3.x compatible	arnold	oscar	major	4.0	3 weeks
#3410	quantization of images truncates instead of rounds	arnold	thiago	major	4.0	3 weeks
#3412	Cook-Torrance MIS functions reflect more light than they receive at near 0 roughness	arnold	alan	major	4.0	3 weeks
#3413	camera exposure not working with semi-opaque regions above the background	arnold	alan	major	4.0	3 weeks
#3415	secondary specular in hair shader missing from specular AOV	arnold	marcos	major	4.0	3 weeks
#3418	rays tangent to a box should return the front facing normal	arnold	thiago	major	4.0	3 weeks
#3423	Scaling double-counted for oriented curves	arnold	mike	major	4.0	2 weeks
#3437	motion blurred quad_light renders incorrectly with enable_fast_lights	arnold	thiago	major	4.0	9 days
#3442	standard material edge darkening	arnold	alan	major	4.0	6 days
#3385	non-deterministic ordering of overlapping primitives across multiple renders	arnold	thiago	minor	4.0	4 weeks
#3449	shadow ray avg. and max hits stat broken	arnold	marcos	minor	4.0	2 days
#3443	remove unused shadow_bias option	arnold	marcos	trivial	4.0	5 days

# 4.0.14.0

## Milestone 4.0.14

### *Enhancements*

- Faster shading in many-light scenes: Quad, disk and spot lights now make much better use of the options.low\_light\_threshold control. In complex scenes with hundreds or thousands of sparse lights that don't all completely overlap, we have seen up to 2x speedups, depending on how aggressively the threshold was tuned. (#2696)
- Improved sampling of rectangular quad lights: Quad area lights now have a solid\_angle parameter that enables a new sampling algorithm that can drastically reduce the variance of samples taken from relatively large or relatively close lights. Currently this technique will not work with irregularly-shaped, non-rectangular quad lights, nor with texture-mapped quad lights, so the option is ignored in these cases, which we hope to fix in a future release. The new option is enabled by default, therefore scenes using big quad area lights should immediately benefit from a reduction in the level of noise. (#889)
- Motion vectors for points primitive: The points primitive can now report motion vectors and time-dependent positional information (supporting the AiShaderGlobalsGetPixelMotionVector() and AiShaderGlobalsGetPositionAtTime() shader utility API functions). Note that the positional information is based on the point centers; since they face the incoming rays, reliable positions cannot be obtained away from the point centers. Generally this should suffice, however. (#3352)
- Interactive geometry creation and destruction: We now allow creation of nodes during an interactive render session, without the need to end the render session and re-create everything from scratch. Interactive destruction of nodes also works, for the simplest cases, but will cause a crash if the node was referenced somehow (from an instance, or when destroying a shader used by an object, etc). Also, procedurals are not destroying all their nodes recursively. These corner cases will be handled in a future release. Trying to destroy an object while rendering will be ignored. Creating a node is done with the usual AiNode() interface, and destroying a node is done via AiNodeDestroy(). (#3371)

### *API additions*

### *Incompatible changes*

- Light beyond low\_light\_threshold is now occluded: Past the low\_light\_threshold we were assuming the light was unoccluded, which tended to reduce the shadow contrast in scenes with many lights and was somewhat counter-intuitive. We now mark it as occluded. This means that past the threshold the image will now be darker. This is more consistent and we have found that more accurate results were produced in all of the scenes that we have tested. (#3363)
- Arnold lib path precedence for Python bindings: For users detaching the Arnold Python bindings from the regular installation folder structure, Python will try to load the Arnold library (libai.so / ai.dll) from multiple locations in order. It will first search in the system library path, and if that doesn't work it then tries in the relative installation folder (which previously was the only search path). (#3116)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3025	Fix metadata parsing issues	arnold	angel	critical	4.0	8 months
#3348	Procedural override does not work correctly when procedural cache is enabled.	arnold	angel	major	4.0	4 weeks
#3356	Propagated opaque attribute not found for shadow rays	arnold	angel	major	4.0	4 weeks
#3359	Updating matrices to an instanced polymesh in IPR causes a crash	arnold	angel	major	4.0	3 weeks
#3362	AiSSSTraceSingleScatter crash due to precision loss	arnold	ramon	major	4.0	3 weeks
#3363	light beyond the low_light_threshold should be occluded	arnold	thiago	major	4.0	3 weeks
#3365	sidedness cannot be overridden in procedurals	arnold	angel	major	4.0	2 weeks
#3366	disk light behavior undefined for shading points behind disk light	arnold	thiago	major	4.0	2 weeks
#3368	MIS not being used if light samples are occluded	arnold	thiago	major	4.0	11 days
#3378	kick -set / .ass input bug	kick	oscar	major	4.0	8 days
#3382	mouse commands in interactive kick mode do not work with keyboard modifiers	arnold	thiago	major	4.0	7 days
#3396	Fix shadow bias for rays coming from under a surface	arnold	ramon	major	4.0	31 hours
#3349	Missing support for escaped characters in metadata files	arnold	angel	minor	4.0	4 weeks

## 4.0.13.1

### Milestone 4.0.13

#### Enhancements

- Lowered polymesh peak memory usage: In previous versions, during the polymesh construction, polymesh memory usage temporarily doubled. This temporary spike could be seen when rendering a few very large meshes. This temporary memory is no longer required which can lower the peak memory usage. (#3330)
- Faster scenes with very many lights: The light acceleration structure has been made more robust for scenes with tens or hundreds of thousands of light sources. In a city scape test scene with 254k lights we observed a 4x speedup. (#2697)
- Faster shadow rays: Shadow rays on average should now be faster to trace. Usually we've seen 3-10% speedups, but we've also seen several shots that became 15% faster. (#3339)
- Reuse importance tables across lights: Texture-mapped quad\_light nodes that point to the same emission shader, and whose resolution parameter is also the same, will now share the precomputed importance table used for efficient importance sampling. This reduces both the startup time and memory usage, specially in scenes with hundreds of lights that previously were constructing a separate importance table each. (#3278)
- Improved sampling of sharp glossy inter-reflections in standard shader: The built-in protection against strong noise (aka fireflies) in glossy inter-reflections has been improved to better deal with very sharp reflections visible through equally sharp reflections. This might result in slightly blurrier/larger secondary highlights, but most of the time this difference won't be perceived. (#3316)
- Improved vector noise: The numerical precision of the vector noise returned by the AiVNoise\*() API calls has been improved. This fixes artifacts in bump and autobump that were reported when using distorted noise relatively far from the origin. Note that this will change the exact shape of the noise, although the overall look and statistical properties will be similar. (#3299)
- Smaller .ass files with sequences of 0's: Long sequences of the commonly-occurring floating point values 0.0 and 1.0 are now encoded more efficiently. We have seen file size reductions of more than 2x in scenes containing volume data. Note that this enhancement is not backwards-compatible: scenes exported with 4.0.13 will not be readable by 4.0.12 or older. (#3314)

#### API additions

- autobump\_visibility: Added a per-object autobump\_visibility mask which can be used to specify what types of rays can perform autobump. This defaults to all rays except for rays that have undergone diffuse bounces. By disabling it for more ray types (like reflection and glossy), rendering can be made faster with little degradation in image quality. (#3132)

#### Incompatible changes

- Differences in vector noise: As mentioned above, the vector noise returned by AiVNoise\*() has changed. For users with already look-dev'ed assets who decide to upgrade in the middle of production, this might be inconvenient. Those users can revert to the older, lower quality vector noise thanks to the new global option enable\_legacy\_vector\_noise. This is set to false by default. Note that this new option is deprecated from the start and will be removed in the next major release. (#3299)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3312	User-data parsing and version checking not thread-safe in Windows	arnold	angel	critical	4.0	6 weeks
#3292	make sure wrap mode "black" wins over "fill" value when they conflict	oioi	ramon	major	4.0	7 weeks
#3300	no need to write "name options" for the options block in .ass files	arnold	marcos	major	4.0	7 weeks
#3307	infinite type lights break with volume_scattering	arnold	thiago	major	4.0	6 weeks
#3310	AiFresnelWeightRGB() does not match AiFresnelWeight()	arnold	alan	major	4.0	6 weeks
#3315	leak in AiMetaDataSetStr when setting the same metadata twice	arnold	angel	major	4.0	5 weeks
#3318	applying a matrix transform to plane results in invalid bounds	arnold	thiago	major	4.0	5 weeks
#3321	NANs in displacement map crash Arnold	arnold	ramon	major	4.0	5 weeks
#3335	Procedural attribute overrides not always preserved when writing to .ass	arnold	angel	major	4.0	4 weeks
#3345	writing out .ass files with tens of thousands of lights is very slow	arnold	marcos	major	4.0	3 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3356	Propagated opaque attribute not found for shadow rays	4.0.13.1	arnold	angel	major	4.0.14
#3362	AiSSSTraceSingleScatter crash due to precision loss	4.0.13.1	arnold	ramon	major	4.0.14

## 4.0.13.0

### Milestone 4.0.13

#### Enhancements

- **Lowered polymesh peak memory usage:** In previous versions, during the polymesh construction, polymesh memory usage temporarily doubled. This temporary spike could be seen when rendering a few very large meshes. This temporary memory is no longer required which can lower the peak memory usage. (#3330)
- **Faster scenes with very many lights:** The light acceleration structure has been made more robust for scenes with tens or hundreds of thousands of light sources. In a city scape test scene with 254k lights we observed a 4x speedup. (#2697)
- **Faster shadow rays:** Shadow rays on average should now be faster to trace. Usually we've seen 3-10% speedups, but we've also seen several shots that became 15% faster. (#3339)
- **Reuse importance tables across lights:** Texture-mapped quad\_light nodes that point to the same emission shader, and whose resolution parameter is also the same, will now share the precomputed importance table used for efficient importance sampling. This reduces both the startup time and memory usage, specially in scenes with hundreds of lights that previously were constructing a separate importance table each. (#3278)
- **Improved sampling of sharp glossy inter-reflections in standard shader:** The built-in protection against strong noise (aka fireflies) in glossy inter-reflections has been improved to better deal with very sharp reflections visible through equally sharp reflections. This might result in slightly blurrier/larger secondary highlights, but most of the time this difference won't be perceived. (#3316)
- **Improved vector noise:** The numerical precision of the vector noise returned by the AiVNoise\*() API calls has been improved. This fixes artifacts in bump and autobump that were reported when using distorted noise relatively far from the origin. Note that this will change the exact shape of the noise, although the overall look and statistical properties will be similar. (#3299)
- **Smaller .ass files with sequences of 0's:** Long sequences of the commonly-occurring floating point values 0.0 and 1.0 are now encoded more efficiently. We have seen file size reductions of more than 2x in scenes containing volume data. Note that this enhancement is not backwards-compatible: scenes exported with 4.0.13 will not be readable by 4.0.12 or older. (#3314)

#### API additions

- **autobump\_visibility:** Added a per-object autobump\_visibility mask which can be used to specify what types of rays can perform autobump. This defaults to all rays except for rays that have undergone diffuse bounces. By disabling it for more ray types (like reflection and glossy), rendering can be made faster with little degradation in image quality. (#3132)

#### Incompatible changes

- **Differences in vector noise:** As mentioned above, the vector noise returned by AiVNoise\*() has changed. For users with already look-dev'ed assets who decide to upgrade in the middle of production, this might be inconvenient. Those users can revert to the older, lower quality vector noise thanks to the new global option enable\_legacy\_vector\_noise. This is set to false by default. Note that this new option is deprecated from the start and will be removed in the next major release. (#3299)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3312	User-data parsing and version checking not thread-safe in Windows	arnold	angel	critical	4.0	3 weeks
#3292	make sure wrap mode "black" wins over "fill" value when they conflict	oii0	ramon	major	4.0	4 weeks
#3300	no need to write "name options" for the options block in .ass files	arnold	marcos	major	4.0	4 weeks
#3307	infinite type lights break with volume_scattering	arnold	thiago	major	4.0	3 weeks
#3310	AiFresnelWeightRGB() does not match AiFresnelWeight()	arnold	alan	major	4.0	3 weeks
#3315	leak in AiMetaDataSetStr when setting the same metadata twice	arnold	angel	major	4.0	2 weeks
#3318	applying a matrix transform to plane results in invalid bounds	arnold	thiago	major	4.0	2 weeks
#3321	NANs in displacement map crash Arnold	arnold	ramon	major	4.0	2 weeks
#3335	Procedural attribute overrides not always preserved when writing to .ass	arnold	angel	major	4.0	8 days
#3345	writing out .ass files with tens of thousands of lights is very slow	arnold	marcos	major	4.0	16 hours

## 4.0.12.0

### Milestone 4.0.12

#### Enhancements

- **Improved numerical robustness:** We have made the renderer more numerically stable in several important cases. First, we have removed shadow artifacts in convex regions that occurred when the object being shaded was far away from the origin. Now it should be much more robust; however, extreme distances from the origin will still breakdown and cause problems due to the limited precision of floating point numbers. This fix also removes the need for setting ray\_bias for shadow rays and in fact ray\_bias is now unused by shadow rays. Second, we have improved the robustness of matrix transformations for static objects far away from the origin. Third, the bump3d shader now uses improved differencing that allow it to work correctly when the shading point is even many thousands of units away from the origin without adjusting bump3d.epsilon, whereas previously the bump effect would start to disappear at even a couple hundreds units away. And fourth, we have improved the encoding of vertex normals at no cost in memory nor speed so that the encoding error is reduced by up to 2x and the vectors round-trip correctly when exported and read again. (#3253, #3281, #3288, #3274)
- **SSS blurring across objects:** It is now possible to tag multiple objects as belonging to the same SSS "set" so that illumination will blur across object boundaries. A common use case might be blurring between teeth and gum geometry. This feature is only available when using raytraced SSS (`sss_bssrdf_samples>0`). It is enabled by adding the constant STRING userdata `sss_setname` to the same value on the objects in the set. (#2413)
- **Improved raytraced SSS in standard shader:** The raytraced SSS technique used by the standard shader has been improved to more robustly handle situations where there is a great difference between the R, G and B color components of the `sss_radius` and/or `Ksss_color` parameters. This can sometimes result in large reductions of noise. Future versions will include a new API so that shader writers can take advantage of this optimization in their own shaders. In addition, a number of small optimizations have been added to the raytraced SSS code which can result in a 5-10% speedup. (#3265)
- **Improved sampling of raytraced gaussian SSS:** Raytraced gaussian SSS diffusion now uses a better importance sampling technique that can significantly reduce noise when compared to the old technique for a given number of samples. (#3266)
- **Per-light scaling of diffuse, specular, SSS:** Three new float parameters have been added to all lights to better control contributions in specific light transport situations: `diffuse`, `specular` and `sss` which can scale the light contribution to each of those independently. These options supersede the now-deprecated boolean light parameters `affect_diffuse` and `affect_specular`, since the new options can be set to zero to disable their effect. Finally, parameters `bounces` and `bounce_factor` have been renamed to `max_bounces` and `indirect` to better reflect their purpose, although the old names still work as synonyms. (#3231)
- **Fresnel from IOR:** A new boolean parameter `Fresnel_use_IOR` has been added to the standard shader, which will calculate Fresnel reflectance based on the IOR parameter, ignoring the values set in `Krn` and `Ksn`. (#3164)
- **Secondary specular in hair shader:** The hair shader has gained a secondary glossy specular lobe, together with the new controls `spec2`, `spec2_color` and `gloss2`. In addition, both specular lobes can now be shifted relative to each other with the new `spec2_shift` and `spec2_shiftcontrols`; usually the first specular lobe is shifted by -5 to -10 degrees, and the secondary lobe is typically `spec2_shift = spec_shift * -1.5` even though they are decoupled for greater artistic control. (#3243)
- **Cheaper Oren-Nayar BRDF:** The math in the Oren-Nayar BRDF has been greatly simplified without changing the look, which may result in a slight speedup. (#3242)
- **Reduced texture I/O for glossy reflections:** Glossy reflections using the Cook-Torrance BRDF will now pull the correct texture mipmap level across all distances while maintaining an optimal amount of texture sharpness. This can result in dramatic reductions in texture data being brought in from disk and lessens the need for using the `texture_glossy.blur` option. In fact, we now recommend setting `texture_glossy.blur` to zero and will likely remove or replace this option in the future. (#3195)
- **texture\_max\_sharpen:** To address some users' concerns that texture lookups are noticeably blurry, we have added the global option `texture_max_sharpen`. It is set to the default of 1.0, which produces the same behavior as before. As it is raised, the textures will appear sharper, but at the expense of increased texture file I/O. The *theoretical* optimum setting for sharpest results is to set this to `AA_samples`, but under most practical situations where texture I/O must be controlled, setting this to around 1.5 already gives sharp results at a moderate cost. See [https://trac.solidangle.com/arnoldpedia/wiki/texture\\_max\\_sharpen](https://trac.solidangle.com/arnoldpedia/wiki/texture_max_sharpen) for more details. (#3184)
- **texertime AOV:** A new built-in AOV called "texertime" has been added which can be used for profiling texture-heavy scenes. This is similar to the "cputime" AOV but instead will produce a float value with the total time (in microseconds) spent on texture access in each pixel. Note that for this AOV to work, the global option `shader_timing_stats` must be enabled first. (#3249)
- **Additional shader timing:** More timings have been added when `options.shader_timing_stats` is enabled which will be printed in the stats section at the end of the log file. It now includes time spent in texture lookups, volume shading, and irradiance caching for pointcloud-based SSS. (#3223)
- **Smaller, faster .ass files:** The storage of .ass files on disk has been optimized by extending base-85 binary encoding to BYTE/INT/UINT arrays, in addition to the already encoded float-based arrays. This can significantly reduce the size of exported .ass files, as well as improve their read and write times. Depending on scene content, you can expect up to 40% smaller files, up to 2x faster reads, and up to 4x faster writes. (#2863)

#### API additions

- **Volume shading API:** A new class of volume shaders is available. These shaders differ from earlier atmosphere shaders in that they are limited to some region of space defined by a container shape, and that heterogenous scattering/absorption effects are now directly integrable by the core instead of volumes having to be ray marched and lit by the shader. These new shaders are point

sampled by the core, provided a small subset of the shader globals (P,Po,Ro,Rd,RI,dPdx,dPdy,M,Minv), and are expected to return their volumetric coefficients via the newly added `AiShaderGlobalsSetVolumeAbsorption()`, `AiShaderGlobalsSetVolumeEmission()` and `AiShaderGlobalsSetVolumeScattering()` API functions. To use this new class of shaders, one must apply them to the shader parameter of a points, sphere or box shape and change the shape's `step_size` parameter to a non-zero value. More information is available on the [arnoldpedia](#). Note: For optimal performance, it is recommended that the step size be no smaller than the smallest feature of the volume, which in the case of voxelized data is the size of a voxel in world space. (#3010,#3011,#3012,#3129,#3155,#3168,#3113)

- **Refraction API:** The previous release, 4.0.11 introduce `AiRefractRay()`. However, that required all refracted rays to use `AiRefractRay()`, otherwise the ray derivatives would remain uninitialized and incorrect results, including NaNs and crashes could occur. We have fixed this so that if `AiRefractRay()` is not called, the same inefficient but at least safe behavior will occur. If `AiRefractRay()` is called after `AiMakeRay()`, then the correct ray derivatives will be computed and performance and texture image quality will be improved. Examples of code usage and results can be seen in <https://trac.solidangle.com/arnoldpedia/wiki/Refracti on>. (#3236)
- **Generic BRDF integrator:** Through the newly added `AiBRDFIntegrate()` function, it is now possible to easily apply the importance sampling Monte Carlo integration technique to compute the indirect lighting for BRDFs defined through the existing MIS API given the `eval_sample`, `eval_brd` and `eval_pdf` callbacks. (#3284)
- **AtRGB operators:** Added arithmetic operators +, -, \*, / for AtRGB similar to those already implemented for AtRGB. (#3233)

## Incompatible changes

None.

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2834	AiMakeRay() does not properly set up shadow rays	arnold	mike	major	3.3	10 months
#3109	Error when append and flush_buckets_on_halt are both true	arnold	ramon	major	4.0	4 months
#3133	Wrong normals when 'ignore_bump' used with autobump	arnold	oscar	major	4.0	4 months
#3147	Filtering long and narrow degenerate ellipses	oilio	ramon	major	4.0	3 months
#3184	textures using a perspective camera are too blurry	arnold	thiago	major	4.0	3 months
#3227	Append is not compatible with crop rendering	arnold	ramon	major	4.0	2 months
#3230	De-clutter logs: min_pixel_width optimization, hair diffuse cache	arnold	marcos	major	4.0	7 weeks
#3234	wrong mipmap level is used for sphere-points with deformation on radius	arnold	thiago	major	4.0	6 weeks
#3236	Nan's in derivatives with refracted rays	arnold	thiago	major	4.0	6 weeks
#3239	NaN's when light's 'indirect' = 0 and 'radius' > 0	arnold	oscar	major	4.0	5 weeks
#3250	Improve precision of time measurements in logged statistics	arnold	angel	major	4.0	4 weeks
#3253	Fix self-shadowing artifacts when far away from the origin	arnold	thiago	major	4.0	4 weeks
#3260	An array with a single element array is not properly written to .ass	arnold	angel	major	4.0	3 weeks
#3261	Arnold crashes when rendering after a texture cache flush	oilio	ramon	major	4.0	3 weeks
#3262	MIS not working with mesh_light if mesh not visible for glossy rays	arnold	ramon	major	4.0	3 weeks
#3263	Incorrect shadow_group override behavior for ginstances	arnold	mike	major	4.0	2 weeks
#3267	Accel structure can create enormous amounts of unaccounted memory	arnold	thiago	major	4.0	2 weeks
#3270	Cannot override "opaque" on procedurals	arnold	angel	major	4.0	2 weeks
#3274	packed normals are slightly wrong	arnold	thiago	major	4.0	9 days
#3281	Improve numerical precision of inv_matrix for static objects	arnold	thiago	major	4.0	7 days
#3288	Make bump3d more robust to scale	arnold	thiago	major	4.0	3 days
#3290	Mismatching matrix motion keys between source and ginstance should be ignored when inherit_xform is off	arnold	mike	major	4.0	3 days
#3295	degenerate motion key causes major slowdown	arnold	thiago	major	4.0	20 hours
#3297	optimize 3 unnecessary divisions in transmittance of standard shader	arnold	marcos	major	4.0	2 hours
#3211	Do not process abort flag when loading procedurals and ass files	arnold	oscar	minor	4.0	2 months
#3240	Mirror uv wrap mode not working for gobo	arnold	ramon	minor	4.0	5 weeks
#3212	Windows 8 is not correctly reported in the OS details of the log file	arnold	oscar	trivial	4.0	2 months

## 4.0.11.0

### Milestone 4.0.11

#### Enhancements

- **Faster processing of big light groups:** Light groups and shadow groups with many lights no longer have as high an overhead to process. In a production shot with thousands of objects and light groups of 300 lights on each object, node initialization time went from 30 secs to 4 secs and total rendering time went down by 4%. (#3148, #3166)
- **Texture lookup cache:** Texture lookups are now cached for each shading point, so redundant texture lookups to the same texture map will not be as expensive. We have seen 20% speedups in certain texture-heavy production shots with complex shader networks. However, it is extremely important that shaderglobals are not reused across shading points otherwise the texture cache will continue to grow with unrelated texture lookups and performance can plummet. If the shading context is set to AI\_CONTEXT\_VOLUME, then the texture lookup cache is not used which allows the sharing of a shaderglobals struct for marching rays across a volume. (#3003)
- **Improved texture derivatives:** Texture derivatives are now properly propagated through reflections and refractions of the polymesh, sphere and point-sphere objects. This improves texture antialiasing and can dramatically reduce disk I/O, provided that the texture maps were already tiled and MIP-mapped. The propagation of texture derivatives for both pointcloud-based and raytraced SSS has also been improved, with the addition of a global option texture\_sss\_blur that lets users trade texture blurring vs I/O. (#2980, #3163, #3180)
- **Faster texture lookups on many-core computers:** We have improved the texture engine's thread scalability, which previously was a bottleneck on highly-threaded renders. On a 32-core machine with 64 logical cores, we saw one render that was not able to scale past 8 threads and actually got slower as more threads were added. By improving the scalability of texture access, especially in the case of string-based lookups, we can now scale to all 64 threads which in this instance gave a 24x speedup for string-based texture lookups when using 64 threads and a 3x speedup for handle-based texture lookups. (#3186, #3190)
- **Faster volume scattering with textured gobos:** Texture lookups during volumetric scattering, e.g. those coming from spotlights with texture mapped gobos, are now much faster, with little to no perceivable difference in shading quality. We have seen 10-20% speedups in simple scenes with a few textured spotlights, depending on how many volume samples are used. (#3181)
- **Auto-detection of max open texture files limit on Linux:** Most Linux distributions have file descriptor limits that are higher than the other platforms by default, but texture\_max\_open\_files was defaulted so low as to not take advantage of this. For scenes with many textures, setting this option higher can result in significant performance gains. The default for Linux is now set higher (512, was 100). If you manually set texture\_max\_open\_files higher than is safe, a warning about potential crashes will be issued. Also, when the option is set to zero Arnold will automatically increase file descriptor limits to the OS-reported hard limit for the process, and then allocate a reasonable chunk of those to textures, usually much larger than 512. Note that you may increase the process descriptor limits yourself on Linux using the ulimit -Sn shell command in bash and limit descriptors in csh/tcsh prior to running kick or another process hosting Arnold, and they will inherit these values for that session. (#3172)
- **Disable memory reporting on Linux:** On older Linux distros, such as CentOS 5, accurately computing the amount of memory used can be very inefficient and in certain situations can amount to a substantial overhead. We have measured this overhead to be up to 10% in complex scenes with long log files. In these situations, it can be beneficial to disable the memory usage calculations by setting the new option, enable\_memory\_reporting, to false. When disabled, memory used will return 0MB and the true peak memory will not be computed. When Arnold terminates, the reported "total peak" memory will not really be the peak, but instead only the memory used at the end of rendering. On CentOS 6 and other recent Linux distros, this option has no effect and memory will continue to be accurately reported even if set to disabled. On Windows and OS X this option does not exist since memory reporting is always fast. (#3159)
- **Bump/autobump time reporting:** To aid with debugging, the time spent in bump mapping and autobump calculations, including the time spent inside the 3 shader calls to the displacement shader, is now reported in the render time stats as long as the new global option shader\_timing\_statsis enabled. There is a bit of overhead (around 1-2%) in the act of measuring itself, which is potentially performed many times per pixel, so we are disabling this option by default. (#3189)
- **New plastic shade mode in utility shader:** In addition to the lambert mode, there is now a plastic shading mode in the utility shader, which has both diffuse (0.7) and specular (0.1) components. The specular component is hardcoded to a Cook-Torrance BRDF with MIS support. This mode can be helpful when debugging and optimizing glossy materials, as it can be quickly assigned to all objects in the scene with the simple kick commands: -is -sm plastic. (#3167)
- **Optimized node iterator for lights:** When creating a node iterator over all the lights in the scene with AiUniverseGetNodeIterator(AI\_NODE\_LIGHT), we now return much faster as we no longer do a linear search over all the (potentially millions of) nodes in the scene. This can help in interactive relighting applications. (#3200)
- **Improved kick -tree:** In addition to an ASCII-art printout of the given shader network, we now include an annotated summary with useful debugging information like maximum depth, total shaders, cycles (if any), shader count by type, and one-to-many connections. (#3187)
- **Upgraded Windows compiler to icc13:** We have upgraded the compiler used to build Arnold on Windows from icc11 to icc13. Initial testing shows speedups in the 5-14% range. (#2986, #3150)
- **Upgraded OIIO to 0.10.15:** This bugfix release supports some rare TIFF extensions from Adobe products (#3122), prints out detailed tile loading statistics per MIP-map level (#3124) and fixes a bug with file wrap mode support (#3064).
- **Upgraded RLM to 9.4BL4:** We have upgraded the license server and the external library controlling the licensing subsystem from version 9.3BL2 to 9.4BL4, a more stable release fixing various crashes, bugs, hangs and memory leaks. (#3111)

#### API additions

- **Refraction ray differentials:** In order to benefit from the refraction ray differentials, a new function, AiRefractRay(), must be called. This will compute the new refracted (or total internal reflection) ray direction as well as compute the proper ray direction differentials. Similarly, AiReflectRay() has been added which computes the reflection direction and ray reflection differentials; however AiMakeRay() is already setting these reflection differentials, so it is not necessary to call this unless the normal has changed, as might happen when reusing an AtRay to cast multiple reflection rays from the same shading point but with different shading normals. Previous code that computed refraction rays would change from:

```
AtRay ray;
AtVector T;
if (AiRefract(&sg->Rd, &sg->Nf, &T, n1, n2))
{
    AiMakeRay(&ray, AI_RAY_REFRACTED, &sg->P, NULL, AI_BIG, sg);
}
else
{ // Total internal reflection
    AtVector R;
    AiReflectSafe(&sg->Rd, &sg->Nf, &sg->Ng, &R);
    AiMakeRay(&ray, AI_RAY_REFLECTED, &sg->P, &R, AI_BIG, sg);
    // But classify it as a refracted ray
    ray.type = AI_RAY_REFRACTED;
}

to:

AtRay ray;
AiMakeRay(&ray, AI_RAY_REFRACTED, &sg->P, NULL, AI_BIG, sg);
AiRefractRay(&ray, &sg->Nf, n1, n2, sg);
```

- **Seeded AtSampler:** A new function, AiSamplerSeeded(), has been added that provides the same functionality as AiSampler() but has an additional integer seed parameter. For shaders and shader networks that use multiple AtSampler instances, each of those would give back the same sample pattern, leading to correlations where the sample patterns were used for very similar effects. Using samplers with unique seeds can help reduce correlations in those cases. (#3131)

## Incompatible changes

- **Removed options.unload\_plugins and procedural.unload\_dso:** The practical utility of these two legacy parameters was dubious and therefore they have now been removed. The default behaviour is preserved: dynamically-loaded libraries (whether for shaders or procedural DSOs) will always be unloaded in AiEnd(). (#3210)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3120	recent changes in abort/interrupt mechanics broke re-rendering	arnold	oscar	critical	4.0	8 weeks
#3199	Empty procedurals cause their library to be reloaded	arnold	oscar	critical	4.0	6 days
#2970	Wrong visibility when instancing procedurals	arnold	angel	major	3.3	5 months
#3018	seams and other artifacts when "blur" is used with textures	oiio	ramon	major	4.0	4 months
#3059	bump nodes don't bridge constant values when 'shader' is not linked	arnold	oscar	major	4.0	3 months
#3117	Support for light nodes in procedurals broken by procedural cache	arnold	angel	major	4.0	2 months
#3122	maketx crashes with TIFF tagged as 16-bit float	oiio	ramon	major	4.0	8 weeks
#3125	diffuse edge darkening in standard shader with Fresnel reflection	arnold	marcos	major	4.0	8 weeks
#3135	min_pixel_width opacity being set too low for points primitives	arnold	thiago	major	4.0	6 weeks
#3136	anisotropic sometimes pulls in overly high-res mip levels	oiio	thiago	major	4.0	6 weeks
#3142	upgrade "degenerate polygons" AI_LOG_DEBUG warnings to AI_LOG_WARNINGS	arnold	thiago	major	4.0	6 weeks
#3143	crash caused by degenerate triangles in displacement	arnold	thiago	major	4.0	6 weeks
#3144	disp_zero_value affecting faces with NULL displacement shaders	arnold	alan	major	4.0	6 weeks
#3146	receive_shadows and self_shadows cannot be overridden in procedurals	arnold	angel	major	4.0	5 weeks
#3148	Lightgroup/shadowgroup membership checks too expensive in scenes with many lights	arnold	thiago	major	4.0	5 weeks
#3154	AI_AOV_BLEND_NONE mode not working for deep AOVs	arnold	ramon	major	4.0	5 weeks
#3156	Wrong disp_padding warnings for motion-blurred meshes	arnold	ramon	major	4.0	4 weeks
#3163	fix texture derivatives for sss_irradiance_shader in pointcloud mode	arnold	mike	major	4.0	4 weeks
#3166	Speed up array simplification during node initialization	arnold	mike	major	4.0	4 weeks
#3174	common radius optimization ignored for points with motion	arnold	thiago	major	4.0	3 weeks
#3176	IPR crash in utility shader when changing shade_mode from ambocc	arnold	angel	major	4.0	3 weeks
#3197	crash with multi-threaded displacement	arnold	alan	major	4.0	8 days
#3207	make ndoteye mode shade black for secondary rays	arnold	marcos	major	4.0	3 days
#3208	perform more thorough error checking when writing ass files	arnold	angel	major	4.0	2 days
#3064	file texture Wrap Mode not working	oiio	ramon	minor	4.0	3 months
#3130	remove annoying kick warnings about null driver when not writing to disk	kick	marcos	minor	4.0	7 weeks
#3188	Arnold hangs when it can no longer write to a tiled EXR file	arnold	ramon	minor	4.0	2 weeks

## 4.0.10.2

### Milestone 4.0.10

#### Enhancements

- **Improved texture blurring for inter-reflections:** We used sg->Rt, the ray type, to determine when to blur a texture with texture\_diffuse.blur and texture\_glossy.blur. We now use the ray depths instead, in the event the ray type changes after a diffuse or glossy bounce. The new method can reduce the amount of texture I/O in inter-reflections, like a glossy sphere reflecting a mirror sphere. (#3057)
- **Faster ray accel build for icc:** An optimization has been made to the ray acceleration structure code that results in build time speedups of these structures of as much as 15%, although typical speedups might be lower depending on the object being built. Furthermore, this fixes a severe performance regression in the icc compiler versions of Arnold that occurred for many types of CPUs (except Sandy Bridge), and in that case build time is as much as 80% faster. (#3099)
- **ignore\_motion.blur discards multiple key data:** When the ignore\_motion.blur option is set to true, or when -imb is passed to kick, we now get rid of the extra motion keys which results in memory savings and faster ray tracing time since we can use the simpler static acceleration structure instead of the one for deformation motion blur. This can be useful for debugging and benchmarking the cost of deformation motion blur in a scene. The previous behavior where the other motion keys are kept can be kept by manually specifying the shutter to be open at only a single point in time -- for instance, -sh 0 0 would render only the first key frame. (#3043)
- **Improved .ass procedural cache:** Cached procedural nodes, i.e. those pointing to the same .ass file on disk, are now directly referenced by one instance node, instead of by a separate instance for each child. This can dramatically reduce the memory footprint in complex scenes, as well as reduce scene loading time. In addition, we have added log stats on how many .ass files have been truly loaded from disk vs found in the cache. Finally, the cache has been extensively and successfully tested in production, so its associated option enable\_procedural\_cache is therefore now enabled by default. (#3054, #3097)
- **Faster resolving of forward references:** In complex scenes made up of many .ass-loading recursive procedurals, resolving forward-referenced symbols like shader names was taking a significant chunk of the initialization time (both pre-, and during rendering, for deferred procedurals). This has been greatly optimized and shouldn't be noticeable any more. (#3051)
- **Faster abort during scene loading:** The renderer will now poll for an abort condition during loading of .ass files and arbitrary procedural nodes, which increases interactivity when working with big procedural/standin networks. (#2568, #3085, #3092)
- **Forced procedural expansion:** A global option procedural\_force\_expand has been added that forces all procedural nodes to be opened before rendering (or before writing to .ass) regardless of whether the user provided bounds for them or not. Note that this effectively makes procedural loading non-deferred, single-threaded and exhaustive; scene loading will generally be slower and use more memory because procedurals will be loaded even when hidden behind other objects. This option is intended mainly as a debugging aid. In addition, a corresponding command-line option for kick has been added: -forceexpand. (#3078)
- **Ignore empty/invalid procedurals:** procedural nodes which produce an error before expansion (e.g. missing procedurals, empty dso parameter, errors while loading) or successfully loaded procedurals that generate no geometry (e.g. empty .ass files, procedurals containing only shaders) were being processed as renderable objects with empty or uninitialized bounding boxes, which caused various internal problems and obscure warnings. These procedurals are now more clearly rejected and never make it into the renderable scene, while also resulting in less warnings. (#3053)
- **Improved max warning/error cutoff:** The maximum number of warnings specified (e.g. kick -nw <n>) now works per warning type and also now applies to errors. Once a particular type of warning/error has been printed N times (the default has been lowered from 200 to 5 in kick), no more of those messages will be shown but other types of messages will still appear. At the end of the log there will be a list of all printed warnings and errors along with the number of times they occurred. (#3066, #3092)
- **Less performance warnings:** We now issue a performance warning for useless identity matrices only when there are more than 100k of them; less than that is probably not worth reporting. (#3069)
- **Enhanced kick attribute overrides:** Extended the -set option for kick to fall back to searching for a node type when a named node is not found. This allows a user to set the same attribute on all nodes of a given type without knowing the node names, which can be very handy, for example kick -set driver\_exr.tiled true. (#3083, #3093)
- **Deep driver improvements:** Some internal infrastructure work has been done in preparation for future support of deep-image output drivers. Threading efficiency has been increased with the addition of an internal non-locking step that allows output drivers to do CPU-intensive work (such as deep-data compression) in parallel without having to wait on other threads. Also, raw drivers can now correctly access all samples in a deep stack, including the first one which was being overwritten and reused for multiple purposes. (#3056, #3068)
- **AiASSWrite() can write to stdout:** In the same way as AiASSLoad("-") reads a scene from stdin, it is now possible to send the scene to stdout with AiASSWrite("-"). (#3118)
- **Faster AiASSWrite() for .ass.gz files:** The zlib compression settings are now optimized for speed instead of file size. The new .ass.gz files are slightly bigger (~5%) but are generated ~3x faster. (#3119)

#### API additions

- **AI\_MAX\_THREADS:** The maximum number of rendering threads is now exposed via the AI\_MAX\_THREADS macro. (#2038)

#### Incompatible changes

- **-resaveop:** Removed obsolete option -resaveop from both kick and pykick. The same result can be obtained by preceding the -resave option with -forceexpand. (#3078)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3087	Multi-threading crash due to uninitialized shaders in procedural networks	arnold	angel	blocker	4.0	5 weeks
#3044	light groups override in procedurals not working	arnold	angel	critical	4.0	8 weeks
#3052	Meshes with identical motion keys and unconnected vertices are corrupted	arnold	thiago	critical	4.0	8 weeks
#3080	Crash when loading deferred .ass procedurals with missing node types	arnold	angel	critical	4.0	6 weeks
#3090	AiTraceProbes crashing with procedurals	arnold	thiago	critical	4.0	5 weeks
#3094	Threading deadlock in .ass-based procedurals due to forward reference resolution	arnold	angel	critical	4.0	5 weeks
#3048	Let volume shaders interact with skydome and distant lights	arnold	ramon	major	4.0	8 weeks
#3056	First deep AOV sample is being overwritten	arnold	ramon	major	4.0	7 weeks
#3065	artifacts in volume scattering with textured spotlight gobos in front of geometry	arnold	oscar	major	4.0	6 weeks
#3072	Fix support for linear basis in curves	arnold	ramon	major	4.0	6 weeks
#3073	Displaced bounding box computation not thread safe	arnold	ramon	major	4.0	6 weeks
#3077	ambient occlusion should be visible to indirect diffuse rays	arnold	marcos	major	4.0	6 weeks
#3082	kick returns error code "1" after rendering if window is open	kick	thiago	major	4.0	6 weeks
#3084	Warning about unsuccessful DLL loading does not give DLL name	arnold	thiago	major	4.0	5 weeks
#3086	visible/unique triangle stats are not correct for nested procedurals	arnold	thiago	major	4.0	5 weeks
#3097	remove excessive "added/found" messages in procedural cache	arnold	oscar	major	4.0	4 weeks
#3104	deferred .ass loading time stats are not thread-safe	arnold	marcos	major	4.0	4 weeks
#3106	.ass-based procedurals should filter out more node types	arnold	marcos	major	4.0	4 weeks
#3110	EXR append does not work when using additional channels besides RGBA	arnold	ramon	major	4.0	3 weeks
#3119	slow writing of compressed .ass.gz files	arnold	angel	major	4.0	13 days
#3061	Minor inconsistencies in node memory stats	arnold	angel	minor	4.0	7 weeks
#3089	rephrase warning for bad assignment of atmosphere shader	arnold	marcos	minor	4.0	5 weeks
#3093	kick crashes on windows when using an invalid -set option	kick	oscar	minor	4.0	5 weeks
#3115	remove procedural indentation in log messages	arnold	marcos	minor	4.0	2 weeks
#3049	Warn when loading procedurals with empty "dso" parameter	arnold	oscar	trivial	4.0	8 weeks
#3081	kick doesn't report the right arnold version	kick	oscar	trivial	4.0	6 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3125	diffuse edge darkening in standard shader with Fresnel reflection	4.0.10.1	arnold	marcos	major	4.0.11

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3120	recent changes in abort/interrupt mechanics broke re-rendering	4.0.10.2	arnold	oscar	critical	4.0.11
#3135	min_pixel_width opacity being set too low for points primitives	4.0.10.2	arnold	thiago	major	4.0.11
#3130	remove annoying kick warnings about null driver when not writing to disk	4.0.10.2	arnold	marcos	minor	4.0.11

## 4.0.10.1

### Milestone 4.0.10

#### Enhancements

- **Improved texture blurring for inter-reflections:** We used sg->Rt, the ray type, to determine when to blur a texture with texture\_diffuse.blur and texture\_glossy.blur. We now use the ray depths instead, in the event the ray type changes after a diffuse or glossy bounce. The new method can reduce the amount of texture I/O in inter-reflections, like a glossy sphere reflecting a mirror sphere. (#3057)
- **Faster ray accel build for icc:** An optimization has been made to the ray acceleration structure code that results in build time speedups of these structures of as much as 15%, although typical speedups might be lower depending on the object being built. Furthermore, this fixes a severe performance regression in the icc compiler versions of Arnold that occurred for many types of CPUs (except Sandy Bridge), and in that case build time is as much as 80% faster. (#3099)
- **ignore\_motion.blur discards multiple key data:** When the ignore\_motion.blur option is set to true, or when -imb is passed to kick, we now get rid of the extra motion keys which results in memory savings and faster ray tracing time since we can use the simpler static acceleration structure instead of the one for deformation motion blur. This can be useful for debugging and benchmarking the cost of deformation motion blur in a scene. The previous behavior where the other motion keys are kept can be kept by manually specifying the shutter to be open at only a single point in time -- for instance, -sh 0 0 would render only the first key frame. (#3043)
- **Improved .ass procedural cache:** Cached procedural nodes, i.e. those pointing to the same .ass file on disk, are now directly referenced by one instance node, instead of by a separate instance for each child. This can dramatically reduce the memory footprint in complex scenes, as well as reduce scene loading time. In addition, we have added log stats on how many .ass files have been truly loaded from disk vs found in the cache. Finally, the cache has been extensively and successfully tested in production, so its associated option enable\_procedural\_cache is therefore now enabled by default. (#3054, #3097)
- **Faster resolving of forward references:** In complex scenes made up of many .ass-loading recursive procedurals, resolving forward-referenced symbols like shader names was taking a significant chunk of the initialization time (both pre-, and during rendering, for deferred procedurals). This has been greatly optimized and shouldn't be noticeable any more. (#3051)
- **Faster abort during scene loading:** The renderer will now poll for an abort condition during loading of .ass files and arbitrary procedural nodes, which increases interactivity when working with big procedural/standin networks. (#2568, #3085, #3092)
- **Forced procedural expansion:** A global option procedural\_force\_expand has been added that forces all procedural nodes to be opened before rendering (or before writing to .ass) regardless of whether the user provided bounds for them or not. Note that this effectively makes procedural loading non-deferred, single-threaded and exhaustive; scene loading will generally be slower and use more memory because procedurals will be loaded even when hidden behind other objects. This option is intended mainly as a debugging aid. In addition, a corresponding command-line option for kick has been added: -forceexpand. (#3078)
- **Ignore empty/invalid procedurals:** procedural nodes which produce an error before expansion (e.g. missing procedurals, empty dso parameter, errors while loading) or successfully loaded procedurals that generate no geometry (e.g. empty .ass files, procedurals containing only shaders) were being processed as renderable objects with empty or uninitialized bounding boxes, which caused various internal problems and obscure warnings. These procedurals are now more clearly rejected and never make it into the renderable scene, while also resulting in less warnings. (#3053)
- **Improved max warning/error cutoff:** The maximum number of warnings specified (e.g. kick -nw <n>) now works per warning type and also now applies to errors. Once a particular type of warning/error has been printed N times (the default has been lowered from 200 to 5 in kick), no more of those messages will be shown but other types of messages will still appear. At the end of the log there will be a list of all printed warnings and errors along with the number of times they occurred. (#3066, #3092)
- **Less performance warnings:** We now issue a performance warning for useless identity matrices only when there are more than 100k of them; less than that is probably not worth reporting. (#3069)
- **Enhanced kick attribute overrides:** Extended the -set option for kick to fall back to searching for a node type when a named node is not found. This allows a user to set the same attribute on all nodes of a given type without knowing the node names, which can be very handy, for example kick -set driver\_exr.tiled true. (#3083, #3093)
- **Deep driver improvements:** Some internal infrastructure work has been done in preparation for future support of deep-image output drivers. Threading efficiency has been increased with the addition of an internal non-locking step that allows output drivers to do CPU-intensive work (such as deep-data compression) in parallel without having to wait on other threads. Also, raw drivers can now correctly access all samples in a deep stack, including the first one which was being overwritten and reused for multiple purposes. (#3056, #3068)
- **AiASSWrite() can write to stdout:** In the same way as AiASSLoad("-") reads a scene from stdin, it is now possible to send the scene to stdout with AiASSWrite("-"). (#3118)
- **Faster AiASSWrite() for .ass.gz files:** The zlib compression settings are now optimized for speed instead of file size. The new .ass.gz files are slightly bigger (~5%) but are generated ~3x faster. (#3119)

#### API additions

- **AI\_MAX\_THREADS:** The maximum number of rendering threads is now exposed via the AI\_MAX\_THREADS macro. (#2038)

#### Incompatible changes

- **-resaveop:** Removed obsolete option -resaveop from both kick and pykick. The same result can be obtained by preceding the -resave option with -forceexpand. (#3078)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3087	Multi-threading crash due to uninitialized shaders in procedural networks	arnold	angel	blocker	4.0	4 weeks
#3044	light groups override in procedurals not working	arnold	angel	critical	4.0	7 weeks
#3052	Meshes with identical motion keys and unconnected vertices are corrupted	arnold	thiago	critical	4.0	6 weeks
#3080	Crash when loading deferred .ass procedurals with missing node types	arnold	angel	critical	4.0	4 weeks
#3090	AiTraceProbes crashing with procedurals	arnold	thiago	critical	4.0	4 weeks
#3094	Threading deadlock in .ass-based procedurals due to forward reference resolution	arnold	angel	critical	4.0	4 weeks
#3048	Let volume shaders interact with skydome and distant lights	arnold	ramon	major	4.0	7 weeks
#3056	First deep AOV sample is being overwritten	arnold	ramon	major	4.0	6 weeks
#3065	artifacts in volume scattering with textured spotlight gobos in front of geometry	arnold	oscar	major	4.0	5 weeks
#3072	Fix support for linear basis in curves	arnold	ramon	major	4.0	5 weeks
#3073	Displaced bounding box computation not thread safe	arnold	ramon	major	4.0	5 weeks
#3077	ambient occlusion should be visible to indirect diffuse rays	arnold	marcos	major	4.0	4 weeks
#3082	kick returns error code "1" after rendering if window is open	kick	thiago	major	4.0	4 weeks
#3084	Warning about unsuccessful DLL loading does not give DLL name	arnold	thiago	major	4.0	4 weeks
#3086	visible/unique triangle stats are not correct for nested procedurals	arnold	thiago	major	4.0	4 weeks
#3097	remove excessive "added/found" messages in procedural cache	arnold	oscar	major	4.0	3 weeks
#3104	deferred .ass loading time stats are not thread-safe	arnold	marcos	major	4.0	2 weeks
#3106	.ass-based procedurals should filter out more node types	arnold	marcos	major	4.0	2 weeks
#3110	EXR append does not work when using additional channels besides RGBA	arnold	ramon	major	4.0	13 days
#3119	slow writing of compressed .ass.gz files	arnold	angel	major	4.0	5 days
#3061	Minor inconsistencies in node memory stats	arnold	angel	minor	4.0	6 weeks
#3089	rephrase warning for bad assignment of atmosphere shader	arnold	marcos	minor	4.0	4 weeks
#3093	kick crashes on windows when using an invalid -set option	kick	oscar	minor	4.0	4 weeks
#3115	remove procedural indentation in log messages	arnold	marcos	minor	4.0	8 days
#3049	Warn when loading procedurals with empty "dso" parameter	arnold	oscar	trivial	4.0	7 weeks
#3081	kick doesn't report the right arnold version	kick	oscar	trivial	4.0	4 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3125	diffuse edge darkening in standard shader with Fresnel reflection	4.0.10.1	arnold	marcos	major	4.0.11

## 4.0.10.0

### Milestone 4.0.10

#### Enhancements

- **Improved texture blurring for inter-reflections:** We used sg->Rt, the ray type, to determine when to blur a texture with texture\_diffuse.blur and texture\_glossy.blur. We now use the ray depths instead, in the event the ray type changes after a diffuse or glossy bounce. The new method can reduce the amount of texture I/O in inter-reflections, like a glossy sphere reflecting a mirror sphere. (#3057)
- **Faster ray accel build for icc:** An optimization has been made to the ray acceleration structure code that results in build time speedups of these structures of as much as 15%, although typical speedups might be lower depending on the object being built. Furthermore, this fixes a severe performance regression in the icc compiler versions of Arnold that occurred for many types of CPUs (except Sandy Bridge), and in that case build time is as much as 80% faster. (#3099)
- **ignore\_motion.blur discards multiple key data:** When the ignore\_motion.blur option is set to true, or when -imb is passed to kick, we now get rid of the extra motion keys which results in memory savings and faster ray tracing time since we can use the simpler static acceleration structure instead of the one for deformation motion blur. This can be useful for debugging and benchmarking the cost of deformation motion blur in a scene. The previous behavior where the other motion keys are kept can be kept by manually specifying the shutter to be open at only a single point in time -- for instance, -sh 0 0 would render only the first key frame. (#3043)
- **Improved .ass procedural cache:** Cached procedural nodes, i.e. those pointing to the same .ass file on disk, are now directly referenced by one instance node, instead of by a separate instance for each child. This can dramatically reduce the memory footprint in complex scenes, as well as reduce scene loading time. In addition, we have added log stats on how many .ass files have been truly loaded from disk vs found in the cache. Finally, the cache has been extensively and successfully tested in production, so its associated option enable\_procedural\_cache is therefore now enabled by default. (#3054, #3097)
- **Faster resolving of forward references:** In complex scenes made up of many .ass-loading recursive procedurals, resolving forward-referenced symbols like shader names was taking a significant chunk of the initialization time (both pre-, and during rendering, for deferred procedurals). This has been greatly optimized and shouldn't be noticeable any more. (#3051)
- **Faster abort during scene loading:** The renderer will now poll for an abort condition during loading of .ass files and arbitrary procedural nodes, which increases interactivity when working with big procedural/standin networks. (#2568, #3085, #3092)
- **Forced procedural expansion:** A global option procedural\_force\_expand has been added that forces all procedural nodes to be opened before rendering (or before writing to .ass) regardless of whether the user provided bounds for them or not. Note that this effectively makes procedural loading non-deferred, single-threaded and exhaustive; scene loading will generally be slower and use more memory because procedurals will be loaded even when hidden behind other objects. This option is intended mainly as a debugging aid. In addition, a corresponding command-line option for kick has been added: -forceexpand. (#3078)
- **Ignore empty/invalid procedurals:** procedural nodes which produce an error before expansion (e.g. missing procedurals, empty dso parameter, errors while loading) or successfully loaded procedurals that generate no geometry (e.g. empty .ass files, procedurals containing only shaders) were being processed as renderable objects with empty or uninitialized bounding boxes, which caused various internal problems and obscure warnings. These procedurals are now more clearly rejected and never make it into the renderable scene, while also resulting in less warnings. (#3053)
- **Improved max warning/error cutoff:** The maximum number of warnings specified (e.g. kick -nw <n>) now works per warning type and also now applies to errors. Once a particular type of warning/error has been printed N times (the default has been lowered from 200 to 5 in kick), no more of those messages will be shown but other types of messages will still appear. At the end of the log there will be a list of all printed warnings and errors along with the number of times they occurred. (#3066, #3092)
- **Less performance warnings:** We now issue a performance warning for useless identity matrices only when there are more than 100k of them; less than that is probably not worth reporting. (#3069)
- **Enhanced kick attribute overrides:** Extended the -set option for kick to fall back to searching for a node type when a named node is not found. This allows a user to set the same attribute on all nodes of a given type without knowing the node names, which can be very handy, for example kick -set driver\_exr.tiled true. (#3083, #3093)
- **Deep driver improvements:** Some internal infrastructure work has been done in preparation for future support of deep-image output drivers. Threading efficiency has been increased with the addition of an internal non-locking step that allows output drivers to do CPU-intensive work (such as deep-data compression) in parallel without having to wait on other threads. Also, raw drivers can now correctly access all samples in a deep stack, including the first one which was being overwritten and reused for multiple purposes. (#3056, #3068)
- **AiASSWrite() can write to stdout:** In the same way as AiASSLoad("-") reads a scene from stdin, it is now possible to send the scene to stdout with AiASSWrite("-"). (#3118)
- **Faster AiASSWrite() for .ass.gz files:** The zlib compression settings are now optimized for speed instead of file size. The new .ass.gz files are slightly bigger (~5%) but are generated ~3x faster. (#3119)

#### API additions

- **AI\_MAX\_THREADS:** The maximum number of rendering threads is now exposed via the AI\_MAX\_THREADS macro. (#2038)

#### Incompatible changes

- **-resaveop:** Removed obsolete option -resaveop from both kick and pykick. The same result can be obtained by preceding the -resave option with -forceexpand. (#3078)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3087	Multi-threading crash due to uninitialized shaders in procedural networks	arnold	angel	blocker	4.0	3 weeks
#3044	light groups override in procedurals not working	arnold	angel	critical	4.0	6 weeks
#3052	Meshes with identical motion keys and unconnected vertices are corrupted	arnold	thiago	critical	4.0	6 weeks
#3080	Crash when loading deferred .ass procedurals with missing node types	arnold	angel	critical	4.0	4 weeks
#3090	AiTraceProbes crashing with procedurals	arnold	thiago	critical	4.0	3 weeks
#3094	Threading deadlock in .ass-based procedurals due to forward reference resolution	arnold	angel	critical	4.0	3 weeks
#3048	Let volume shaders interact with skydome and distant lights	arnold	ramon	major	4.0	6 weeks
#3056	First deep AOV sample is being overwritten	arnold	ramon	major	4.0	6 weeks
#3065	artifacts in volume scattering with textured spotlight gobos in front of geometry	arnold	oscar	major	4.0	5 weeks
#3072	Fix support for linear basis in curves	arnold	ramon	major	4.0	4 weeks
#3073	Displaced bounding box computation not thread safe	arnold	ramon	major	4.0	4 weeks
#3077	ambient occlusion should be visible to indirect diffuse rays	arnold	marcos	major	4.0	4 weeks
#3082	kick returns error code "1" after rendering if window is open	kick	thiago	major	4.0	4 weeks
#3084	Warning about unsuccessful DLL loading does not give DLL name	arnold	thiago	major	4.0	4 weeks
#3086	visible/unique triangle stats are not correct for nested procedurals	arnold	thiago	major	4.0	3 weeks
#3097	remove excessive "added/found" messages in procedural cache	arnold	oscar	major	4.0	3 weeks
#3104	deferred .ass loading time stats are not thread-safe	arnold	marcos	major	4.0	2 weeks
#3106	.ass-based procedurals should filter out more node types	arnold	marcos	major	4.0	2 weeks
#3110	EXR append does not work when using additional channels besides RGBA	arnold	ramon	major	4.0	10 days
#3119	slow writing of compressed .ass.gz files	arnold	angel	major	4.0	36 hours
#3061	Minor inconsistencies in node memory stats	arnold	angel	minor	4.0	5 weeks
#3089	rephrase warning for bad assignment of atmosphere shader	arnold	marcos	minor	4.0	3 weeks
#3093	kick crashes on windows when using an invalid -set option	kick	oscar	minor	4.0	3 weeks
#3115	remove procedural indentation in log messages	arnold	marcos	minor	4.0	4 days
#3049	Warn when loading procedurals with empty "dso" parameter	arnold	oscar	trivial	4.0	6 weeks
#3081	kick doesn't report the right arnold version	kick	oscar	trivial	4.0	4 weeks

## 4.0.9.1

### Milestone 4.0.9

#### Enhancements

- **Reduced texture I/O for glossy reflection:** For rays of type AI\_RAY\_GLOSSY, the differentials used to calculate texture sampling footprints have been improved resulting in generally larger filter sizes while still matching the requested visible detail. Texture sampling will now choose less detailed MIP levels when possible and will generally load less data from disk and do less work. (#3034)
- **Faster missing textures:** For handle-based texture access via AiTextureHandleAccess(), the texture subsystem will not be redundantly queried once we have determined that a handle corresponds to an unreadable file. This can improve performance in scenes with many missing textures, which surprisingly can happen in certain production pipelines. (#3040)

#### API additions

#### Incompatible changes

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3032	Statically link OIIO with jpeg/tiff/png/exr/boost in OSX	oiio	ramon	blocker	4.0	2 weeks
#3037	OIIO failed assertion with accumulated error messages	arnold	mike	blocker	4.0	13 days
#3035	fix "unspecified OIIO error" message for missing EXR textures	oiio	ramon	critical	4.0	2 weeks
#3041	Do not print empty OIIO error messages	arnold	ramon	critical	4.0	10 days
#2853	Crash when a texture map has non-power of two tiles	oiio	ramon	major	4.0	4 months
#2974	remove annoying log messages about unresolved tags	arnold	angel	major	3.3	7 weeks
#3026	cannot create packed arrays with more than about 1 billion elements	arnold	thiago	major	4.0	3 weeks
#3034	Compute ray direction differentials for glossy reflection	arnold	marcos	major	4.0	2 weeks
#3050	memory leak in AI_RENDER_MODE_FREE	arnold	thiago	major	4.0	7 days
#3036	reporting wrong number of keys in bvh_motion build	arnold	thiago	minor	4.0	13 days

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#3052	Meshes with identical motion keys and unconnected vertices are corrupted	4.0.9.1	arnold	thiago	critical	4.0.10
#3048	Let volume shaders interact with skydome and distant lights	framestore, 4.0.9.1	arnold	ramon	major	4.0.10
#3049	Warn when loading procedurals with empty "dso" parameter	4.0.9.1	arnold	oscar	trivial	4.0.10

# 4.0.9.0

## Milestone 4.0.9

### Enhancements

- **Reduced texture I/O for glossy reflection:** For rays of type AI\_RAY\_GLOSSY, the differentials used to calculate texture sampling footprints have been improved resulting in generally larger filter sizes while still matching the requested visible detail. Texture sampling will now choose less detailed MIP levels when possible and will generally load less data from disk and do less work. (#3034)
- **Faster missing textures:** For handle-based texture access via AiTextureHandleAccess(), the texture subsystem will not be redundantly queried once we have determined that a handle corresponds to an unreadable file. This can improve performance in scenes with many missing textures, which surprisingly can happen in certain production pipelines. (#3040)

### API additions

### Incompatible changes

### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#3032	Statically link OIIO with jpeg/tiff/png/exr/boost in OSX	oiio	ramon	blocker	4.0	7 days
#3037	OIIO failed assertion with accumulated error messages	arnold	mike	blocker	4.0	6 days
#3035	fix "unspecified OIIO error" message for missing EXR textures	oiio	ramon	critical	4.0	7 days
#3041	Do not print empty OIIO error messages	arnold	ramon	critical	4.0	3 days
#2853	Crash when a texture map has non-power of two tiles	oiio	ramon	major	4.0	4 months
#2974	remove annoying log messages about unresolved tags	arnold	angel	major	3.3	6 weeks
#3026	cannot create packed arrays with more than about 1 billion elements	arnold	thiago	major	4.0	2 weeks
#3034	Compute ray direction differentials for glossy reflection	arnold	marcos	major	4.0	7 days
#3050	memory leak in AI_RENDER_MODE_FREE	arnold	thiago	major	4.0	3 hours
#3036	reporting wrong number of keys in bvh_motion build	arnold	thiago	minor	4.0	6 days

## 4.0.8.0

### Milestone 4.0.8

#### Enhancements

- **Reduced texture I/O:** Ray differentials, used to calculate texture sampling footprints, have been improved resulting in generally larger filter sizes matching the requested visible detail. Texture sampling will now choose less detailed MIP levels when possible and will generally load less data from disk and do less work. In particular, ray differentials have been improved for mirror reflection rays (of type AI\_RAY\_REFLECTED), refraction rays (in the standard shader only), ray-traced SSS rays, and finally for camera rays during the first rough passes of progressive rendering at negative AA settings, where textures will be blurred to match the size of the blocks of pixels and help decrease startup time. We have seen dramatic speedups in scenes with large numbers of high-resolution texture maps because of the significantly reduced disk I/O. (#2979, #3017, #3022, #3028)
- **Faster bump mapping:** Autobump and bump mapping are now cheaper to compute than before due to there being 25% less shader evaluations per call. (#2909)
- **Faster ray accel build:** It now takes about 25% less time to build the ray acceleration structures. Memory usage and ray tracing speed are unaffected. (#3008)
- **Thread affinity:** Threads can now be pinned to cores on Linux. This can improve scalability in modern machines with 16 or 24+ threads and, in the case of ray acceleration structure builds, it can occasionally result in as much as a 3x speedup when all the cores are being used (which is the default in Arnold, options.threads = 0 or -t 0 in kick). The default setting is auto where thread pinning is only enabled if more than half the logical cores are being used. This can also be manually set to always on or off by setting the new global option options.pin\_threads to on, off, or auto. Note that, if client code, for instance a custom shader, spawns their own threads manually (with pthread\_create or similar), these threads will inherit the same thread affinity, which totally breaks the point of spawning threads; in these situations they can either setoptions.pin\_threads to off or they can create their threads with the Arnold API AIThreadCreate() which will un-pin the created thread. (#2733)
- **Auto-instancing of .ass procedurals:** Thanks to a new built-in cache, procedural nodes whose dso parameter points to an .ass file can be automatically cached for future reuse; subsequent procedurals trying to load the same .ass file will instance the previously loaded geometry, potentially saving tons of both memory and disk I/O. Because of the experimental nature of this feature, we have opted for disabling it by default and let those users that need it manually enable it with the global option enable\_procedural\_cache. (#2937)
- **AiOrenNayarIntegrate() caching for shader networks:** Just like we already do with AiIndirectDiffuse(), multiple calls to AiOrenNayarIntegrate() with the same arguments within a shader network will now be optimized so that the result is only calculated once and then reused. (#3004)
- **bumpdiff mode in the utility shader:** The utility shader has a new value available for the color\_mode parameter: bumpdiff. This will show how far the bump and autobump normals vary from the base smooth-shaded normals as a heatmap (blue is the same, going through green to red as varying up to 90 degrees away). This is useful for debugging the balance between subdivision iterations with displacement vs autobump making up for the rest. (#3002)
- **Log file improvements and cleanups:** A number of minor annoyances in .log files have been fixed, and a number of messages have been made more helpful. For example: the "ray counts" section now mention the number of rays fired from SSS pointclouds which were previously not reported; the "render time" section now reports the time it takes to build the importance sampling tables for the skydome\_light and quad\_light nodes; the "subdivision done" message now more clearly reports the initial number of polygons and the final number of subdivided quads; the "displacement done" message now reports the number of vertices that were displaced; and the "loading metadata file" message now only appears with debug verbosity. (#3029)
- **Updated pykick:** The Python version of kick has been updated so that it now has all the recent command-line options that were added in the C version of kick. (#2984)
- **Static RLM linking in Windows:** To make things more consistent with Linux and OSX, we now statically link the RLM library in Windows too; we don't distribute the files rlm\*.dll anymore. (#2982)

#### API additions

- **AiUDataGetNode():** User-data queries of type NODE are now supported. Recall that the NODE type is very similar to the POINTER type, with the only difference that an element of NODE type is guaranteed to point to an Arnold AtNode object rather than being a pointer to an arbitrary memory location. (#2771)

#### Incompatible changes

- **Changed AiLicenseGetInfo():** The AiLicenseGetInfo API has been changed so that it doesn't use std::vector, which is known to cause problems across compilers and compiler settings. The new safer API is: (#2826)

```
AI_API int AiLicenseGetInfo(AtLicenseInfo*& licenses, unsigned int& n)
```
- **Removed ARNOLD\_DISPLACEMENT\_DERIVS\_HACK:** This undocumented legacy environment variable is no longer necessary. (#2586)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2953	shading artifacts in procedural networks with instances of procedurals	arnold	oscar	critical	4.0	6 weeks
#2842	Arnold crashing on unexpected camera shader	arnold	oscar	major	4.0	4 months
#2979	Compute ray direction differentials for reflection	arnold	mike	major	4.0	4 weeks
#2983	faceted artifacts in raytraced SSS	arnold	alan	major	4.0	4 weeks
#2985	Crash in AiEnd when using AiRender after an AiASSWrite	arnold	oscar	major	4.0	4 weeks
#2987	Crash when aborting on license fail	arnold	oscar	major	4.0	4 weeks
#2989	AiSampler() crashes when sample count is 0	arnold	alan	major	4.0	4 weeks
#2993	Tile boundaries incorrect for udim tag	arnold	ramon	major	4.0	3 weeks
#2994	OS X compiled shaders do not load	arnold	ramon	major	4.0	3 weeks
#2997	Distant lights do not cast shadow rays in volumes	arnold	ramon	major	4.0	3 weeks
#2998	random corruption of rays	arnold	alan	major	4.0	3 weeks
#3000	Improve numerical robustness of texture derivatives	arnold	thiago	major	4.0	3 weeks
#3014	missing Doxygen docs for AiASSWrite/AiASSLoad	arnold	marcos	major	4.0	2 weeks
#3016	broken normal smoothing for NURBS primitives	arnold	oscar	major	4.0	2 weeks
#3017	Compute ray direction differentials for refraction in standard shader	arnold	mike	major	4.0	13 days
#3021	correlation artifact in indirectly sampled area lights	arnold	alan	major	4.0	9 days
#3027	"ray counts" stats are missing SSS rays	arnold	mike	major	4.0	5 days
#2977	Memory leak in kick when setting the outputs	kick	oscar	minor	4.0	4 weeks
#2996	Texture Errors Should Report Filename	arnold	ramon	minor	4.0	3 weeks

# 4.0.7.0

## Milestone 4.0.7

### Enhancements

- **Ray-traced BSSRDFs:** A new alternative to point-cloud subsurface scattering has been added. This method is brute-force ray-tracing based, but produces visually identical results to the pointcloud technique. This approach removes a number of shortcomings of the pointcloud method, trading them for unbiased noise instead. The new method consumes zero extra memory, supports motion blur, supports interactive relighting, is fully multi-threaded, starts up immediately and does not degrade in performance as the scattering radius shrinks. The new method is enabled by the global option: `sss_bssrdf_samples`. This option defaults to 0 for backwards compatibility (uses pointclouds). When set to a value greater than 0, this number of lighting samples (direct and indirect) will be taken to estimate lighting within a radius of the point being shaded. On top of the direct/indirect irradiance rays, the algorithm uses short probe rays of type `AI_RAY_DIFFUSE` to find points near the one being shaded, so this can bias the ray statistics in the .log file a bit. (trac#2763)
- **Improved sampling at secondary bounces:** The 2D sampling patterns generated by Arnold will now be stratified even for secondary bounces instead of reverting to fully random sampling as before. This can greatly reduce the amount of noise, particularly when rendering things that are seen through near-specular effects like glossy reflections or refractions. (#2371)
- **Improved sampling for SSS Point cloud:** When baking diffuse lighting calculations into the SSS pointcloud, we now use MIS to reduce noise from large nearby area lights and the skydome. This may increase the number of rays slightly, but give higher quality sampling in those cases. (trac#2484)
- **New mesh\_light:** There is a new type of light available called `mesh_light` which takes a polymesh as a parameter and emits light using the polymesh's surface geometry. Note: The mesh light's matrix transform is being ignored for the time being and only the linked mesh's transformation will be taken into account. (#2864)
- **Improved smoothed normals:** The quality of the smooth shading normals stored per-vertex in triangular tessellations has been improved by adjusting the weights used to average the triangle normals around a vertex. In addition to triangle area, the weights now factor the angle between the two triangle edges containing the shared vertex. This will be most evident for coarse meshes where the face normal and the smoothed vertex normals substantially differ. (#2923)
- **Mari UDIM texture tags:** The built-in image node now recognizes Mari UDIM tags in texture filenames and substitutes in the proper values. A vanilla tag, e.g. `<udim>`, and customized dimensions, e.g. `<udim:100>` are recognized. Please note that these will disable texture handles and may impact performance; we hope to improve this in a future release. (#2670)
- **autobump shader call stats:** The number of autobump shader calls is now properly reported in the log statistics. In previous versions, this could "hide" a large number of shader evaluations, since autobump computes the bump differentials by calling the displacement shader 4 times, which can quickly become the dominant source of shader evaluations during rendering. (#2899)
- **New ignore\_dof global option:** Analogously to `ignore_motion_blur`, there is now an `ignore_dof` option which lets you quickly deactivate depth of field effects in the cameras that support it. This is equivalent to setting the `aperture_size` to 0 in the camera. In addition, an `-idof` command-line argument has been added to kick for convenience. (#2947)
- **Find DSO procedurals with kick -nodes:** In addition to listing the dynamically-loaded shaders, `kick -nodes` now lists any geometry-generation DSO/DLL's used in procedural nodes (aka standins) that are found in ".", `ARNOLD_PLUGIN_PATH` or any other searchpaths specified in the `kick` command-line via the `-I <path>` option. (#2942)
- **Improved stack traces:** In the event of a crash, the stack trace (or "backtrace") is now reported in more detail, specially in debug builds for developers. Also, backtraces are now implemented in Windows for the first time. (#2827, #1739)

### API additions

None.

### Incompatible changes

None.

### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2930	support for Windows XP was accidentally broken in Arnold 4.0.6.0	arnold	oscar	blocker	4.0	4 weeks
#2627	monochrome images with alpha not read correctly	oioi	ramon	major	3.3	7 months
#2860	Free-floating mesh vertices crash subdivision	arnold	mike	major	3.3	2 months
#2904	Improve numerical precision of subdivision limit normals	arnold	thiago	major	3.3	6 weeks
#2905	Don't print escape sequences (colors) in logs if stdout is redirected	arnold	oscar	major	4.0	6 weeks
#2907	Oriented curves with identical motion keys corrupts orientations	arnold	mike	major	3.3	6 weeks
#2908	Crash with pixel-width optimization and constant radius specification	arnold	oscar	major	4.0	6 weeks
#2911	Crash when EXR/TIFF output cannot be opened in 'append' mode	arnold	ramon	major	3.3	5 weeks
#2912	Race condition in latent deferred SSS sample code causes random crashes	arnold	oscar	major	4.0	5 weeks
#2915	.ass parser reports incorrect line numbers when a string contains EOL's	arnold	angel	major	3.3	5 weeks
#2916	Arnold doesn't respond correctly to Ctrl+C events in windows	arnold	oscar	major	4.0	5 weeks
#2917	quantization artifacts in smooth edge of spot_light	arnold	oscar	major	4.0	5 weeks
#2921	AtNodeEntry counters are not always atomically changed	arnold	mike	major	3.3	5 weeks
#2925	maketx quantization errors when rescaling 8- and 16-bit images	oioi	ramon	major	3.3	4 weeks
#2938	ginstance attributes with default value are all exported to .ass	arnold	angel	major	3.3	3 weeks
#2941	write out floats using all bits of precision	arnold	thiago	major	3.3	3 weeks
#2943	fix warning when shader_searchpath points to a nonexisting path	arnold	marcos	major	3.3	2 weeks
#2948	Nodes with a "@before" link are always exported to .ass file as ginstance	arnold	angel	major	3.3	13 days
#2949	bug when tessellating a single polygon with a simulated hole	arnold	oscar	major	4.0	13 days
#2952	Fix run-time search paths hard-coded in "ai" and "kick"	arnold	oscar	major	4.0	12 days
#2960	incorrect user bounds warning when bounds are equal.	arnold	thiago	major	3.3	12 days
#2961	Crash with NaNs in displacement map	arnold	oscar	major	4.0	11 days
#2964	Missing ray type defines in Python API	arnold	angel	major	3.3	8 days
#2965	Empty error message when accessing a non-existent EXR texture	oioi	oscar	major	4.0	7 days
#2702	IOR set to 0 in the standard shader gives a fatal error	arnold	marcos	minor	3.3	5 months
#2933	improved reporting of light samples in the log file	arnold	marcos	minor	3.3	3 weeks
#2934	Recover extended OS info (compatible with windows XP)	arnold	oscar	minor	4.0	3 weeks

# 4.0.6.0

## Milestone 4.0.6

### Enhancements

- **Improved thread scaling in texture lookups:** Fixed a major slowdown that occurred when performing many texture lookups per sample. This was most evident on multi-socket systems. On a dual hexa-core system (24 threads) this resulted in a 16x speedup for a scene with more than a thousand texture lookups per pixel. We have seen 2x and 4x speedups in other scenes. The actual speedup will depend on various things like the hardware, the number of texture lookups per shader network, and on the size of the texture cache, which is 512 MB by default. (#2887)
- **Subdivision optimizations:** Subdivision now uses substantially less memory and takes slightly less time to compute. In addition, the size of the AtNode structure has been reduced by 8 bytes, which will help save some memory in scenes with millions of instances. (#2835, #2888)
- **Improved ray triangle intersection precision:** The numerical robustness of the ray/triangle test has been improved, fixing some long-standing issues with rays slipping through edges. While some errors are still theoretically possible, the probability is dramatically reduced. Render times should also improve; we've seen up to 7% improvements, but in practice it is likely to just be a couple percent. The global render option enable\_hit\_refinement and the per-object option double\_precision\_intersector have both been removed. Hit refinement is now hardcoded for camera rays. (#2839, #2862)
- **Camera exposure control:** An exposure parameter has been added to all cameras, even user-written ones, which scales the pixel samples by  $2^{\text{exposure}}$ . Increasing the exposure by 1.0, or one "stop", produces an image twice as bright, while reducing the exposure by one stop produces an image half as bright. The default value is 0 for backwards compatibility. This can also be controlled in kick with the new -e <f> option. (#2884)
- **PLY support:** PLY files in both ASCII and binary variants can be loaded as procedurals, similar to OBJ files. In addition to basic polygon data, per-vertex normals are also supported. (#1866)
- **New id color mode in utility shader:** The id mode can be used to visualize a different color for each object. Unlike the obj mode, which is based on a hash of the object's name string, the id mode is based on a hash of the object's id integer attribute and is therefore more stable in animation in cases where the object name changes per frame (because of e.g. a frame number suffix). (#2086)
- **OS details in logs:** Details about the running operating system are now printed at the top of the log file, such as the name, version, codename, kernel, etc (depending on the OS type). (#2734)

```
00:00:00      5MB      |  running on mos
00:00:00      5MB      |  1 x Intel(R) Core(TM) i7-2860QM CPU @ 2.50GHz (4 cores, 8 logical)
with 16384MB
00:00:00      5MB      |  Mac OS X 10.7.3 "Lion" (build 11D50b), darwin kernel 11.3.0
```

- **Log file specification for kick:** kick now takes an optional -logfile parameter on the command line to write out the log to a specific location, rather than the default. This supercedes the existing -log option which does not take a parameter. (#2878)

```
-logfile <s>      Enable log file and write to the specified file path
```

### API additions

- **AiNodeSetAttributes():** Multiple attributes of an AtNode can now be set in one go by using a string which follows the same format rules for setting node parameters as used in .ass files. This can be useful in the Maya/Softimage/Katana plugins to pass new options to the renderer before they are made visible in the UI of the host application. (#2531)

```
AI_API void AiNodeSetAttributes(AtNode* node, const char* attributes);
```

### Incompatible changes

None.

### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2887	Thread contention with many texture lookups per sample on multi-socket machines	arnold	mike	blocker	3.3	10 days
#2096	Fix gamma correction dependency on node creation order	arnold	angel	major	3.3	17 months
#2838	popping artifacts in multi-threaded displacement	arnold	oscar	major	3.3	6 weeks
#2844	Wrong gamma correction on binary encoded arrays	arnold	angel	major	3.3	5 weeks
#2846	Orientations ignored for curves when not in oriented mode	arnold	mike	major	3.3	5 weeks
#2850	Cannot switch to/from oriented curves mode in IPR	arnold	mike	major	3.3	5 weeks
#2854	UV-based displacement combined with deformation motion blur broken	arnold	mike	major	3.3	4 weeks
#2856	Random slowness with curves intersections	arnold	mike	major	3.3	4 weeks
#2857	Auto setting of options->camera not working after deleting previous camera	arnold	angel	major	3.3	4 weeks
#2865	Shader gamma is not applied to params with the default value	arnold	angel	major	3.3	3 weeks
#2870	stack overflow when parsing .ass files (arrays of arrays)	arnold	angel	major	3.3	3 weeks
#2872	AiLightsGetShadowMatte() always black after a light loop	arnold	alan	major	3.3	2 weeks
#2877	temporary memory leak in AiTraceProbe()	arnold	thiago	major	3.3	2 weeks
#2883	image shader creates texture handles even if cache_texture_handles is off	arnold	angel	major	3.3	2 weeks
#2894	Ray Differentials not set when a ray only hits background	arnold	ramon	major	3.3	2 days
#2895	AiMappingLatLongDerivs returns wrong derivatives	arnold	ramon	major	3.3	31 hours
#2896	Shadow rays cast with AiTrace do not skip shaders on opaque hits	arnold	mike	major	3.3	23 hours
#2897	shader calls stats missing evaluations inside shader networks	arnold	marcos	major	3.3	19 hours
#2867	fix documentation for AiShaderGlobalsEvaluateBump()	arnold	mike	minor	3.3	3 weeks

## 4.0.5.3

### Milestone 4.0.5

#### Enhancements

- **Multi-threaded ray accel build for deformation motion blur:** Similar to the 4.0.3 release (#2653) that added support for building non-motion blurred objects in parallel, we now allow for multiple render threads to build a motion blurred object's ray accel in parallel. On a 12 core machine this lets us build the accel structure for a hair object 9.5x faster. (#2755)
- **Miscellaneous ray accel and subdivision optimizations:** The ray accel structure for deformation motion blurred objects now uses 7% less memory. Ray accel structure builds, especially when using many threads, will now be a percent or two faster. Triangulation is now roughly 2x faster; as this is also used at the end of the Catmull-Clark subdivision process (which generates quads that need to be triangulated), subdivision will also be slightly faster, around 10-15% faster overall. (#2816, #2811, #2817)
- **Multi-threaded .ass loading:** The .ass parsing code has been rewritten and made thread-safe, which allows multiple .ass-based procedurals to be processed in parallel, as it already happened with DSO and funcptr procedurals. If needed, parallel loading can be disabled with options.enable\_threaded\_procedurals. (#2113)
- **Oriented curves:** The curves node now has an additional mode oriented that allows the specification of orientations via the orientations vector array. It behaves similar to the ribbon mode, except curves can face away from the camera or incoming ray according to the orientations; curves also get thinner when they face away. Uses include grass, wider ribbons, or even leaves. Orientations are specified per-point, the same number as there are control points. (#797, #2824)
- **Accurate memory reporting on Linux:** The memory usage on Linux is now correctly reported. The older code was reporting virtual memory which is significantly larger than what is really being used. The new code reports both the memory being used on physical RAM (resident memory) as well as the memory in swap. This enhanced reporting has some overhead on older Linux distros, so scenes with thousands of objects and/or log messages might take a few hundred milliseconds longer. On at least Linux kernel 2.6.34 or RHEL6/CentOS6 and newer there is no overhead. (#2602)
- **Checkpointing renders:** EXR and TIFF drivers now support render checkpointing (or "append mode"). By setting the corresponding output driver's .append attribute to true Arnold will preserve previously rendered tiles and only work on the missing ones, appending them to the output files. If no image is present the render will proceed as normal creating a new image. If image specifications do not match the render will be aborted. (#2417)
- **Deferred .ass parsing time in log stats:** The .ass parsing time reported in the log statistics is now split into two categories: parsing when loading the main scene prior to rendering, and parsing of deferred-loading .ass procedurals during rendering. (#2795)
- **Enhanced backtrace dumps:** Backtrace dumps for crashes in debug builds are now cleaner and show more helpful information, along with colored output and time of the crash. If available, it will also report the pixel location where the crash occurred. (#2716, #2717, #2831, #2832)
- **Added console colors to log messages in Windows:** The log output sent to the console will now use colors to highlight warnings in yellow and errors in red. This feature was already available in Linux/OSX but was missing in Windows. (#2830)

#### API additions

#### Incompatible changes

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2602	Cleanup Linux memory reporting	arnold	thiago	major	3.3	5 months
#2797	Running out of memory with high ray depth	arnold	thiago	major	3.3	6 weeks
#2798	Crash writing bump2d to .ass with a value set to the "shader" parameter	arnold	angel	major	3.3	6 weeks
#2800	Bug in Python API functions returning a pointer	arnold	angel	major	3.3	5 weeks
#2812	OIIO failed assertion with accumulated error messages	arnold	ramon	major	3.3	5 weeks
#2818	memory leak when closing plugin file handle on linux/os x	arnold	thiago	major	3.3	4 weeks
#2820	NANs in Ward-Duer BRDF with subdiv_smooth_derivs enabled	arnold	alan	major	3.3	4 weeks
#2821	rare crash in Ward-Duer BRDF when used with volume_scattering	arnold	alan	major	3.3	4 weeks
#2718	Long multi-paths are cut out with "kick -l"	arnold	oscar	minor	3.3	3 months
#2803	"included" .ass files should be taken into account in ".ass parsing" time	arnold	oscar	minor	3.3	5 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2844	Wrong gamma correction on binary encoded arrays	4.0.5.1	arnold	angel	major	4.0.6
#2854	UV-based displacement combined with deformation motion blur broken	4.0.5.1	arnold	mike	major	4.0.6

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2838	popping artifacts in multi-threaded displacement	framestore, 4.0.5.2	arnold	oscar	major	4.0.6

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2856	Random slowness with curves intersections	4.0.5.3	arnold	mike	major	4.0.6

## 4.0.5.2

### Milestone 4.0.5

#### Enhancements

- **Multi-threaded ray accel build for deformation motion blur:** Similar to the 4.0.3 release (#2653) that added support for building non-motion blurred objects in parallel, we now allow for multiple render threads to build a motion blurred object's ray accel in parallel. On a 12 core machine this lets us build the accel structure for a hair object 9.5x faster. (#2755)
- **Miscellaneous ray accel and subdivision optimizations:** The ray accel structure for deformation motion blurred objects now uses 7% less memory. Ray accel structure builds, especially when using many threads, will now be a percent or two faster. Triangulation is now roughly 2x faster; as this is also used at the end of the Catmull-Clark subdivision process (which generates quads that need to be triangulated), subdivision will also be slightly faster, around 10-15% faster overall. (#2816, #2811, #2817)
- **Multi-threaded .ass loading:** The .ass parsing code has been rewritten and made thread-safe, which allows multiple .ass-based procedurals to be processed in parallel, as it already happened with DSO and funcptr procedurals. If needed, parallel loading can be disabled with options.enable\_threaded\_procedurals. (#2113)
- **Oriented curves:** The curves node now has an additional mode oriented that allows the specification of orientations via the orientations vector array. It behaves similar to the ribbon mode, except curves can face away from the camera or incoming ray according to the orientations; curves also get thinner when they face away. Uses include grass, wider ribbons, or even leaves. Orientations are specified per-point, the same number as there are control points. (#797, #2824)
- **Accurate memory reporting on Linux:** The memory usage on Linux is now correctly reported. The older code was reporting virtual memory which is significantly larger than what is really being used. The new code reports both the memory being used on physical RAM (resident memory) as well as the memory in swap. This enhanced reporting has some overhead on older Linux distros, so scenes with thousands of objects and/or log messages might take a few hundred milliseconds longer. On at least Linux kernel 2.6.34 or RHEL6/CentOS6 and newer there is no overhead. (#2602)
- **Checkpointing renders:** EXR and TIFF drivers now support render checkpointing (or "append mode"). By setting the corresponding output driver's .append attribute to true Arnold will preserve previously rendered tiles and only work on the missing ones, appending them to the output files. If no image is present the render will proceed as normal creating a new image. If image specifications do not match the render will be aborted. (#2417)
- **Deferred .ass parsing time in log stats:** The .ass parsing time reported in the log statistics is now split into two categories: parsing when loading the main scene prior to rendering, and parsing of deferred-loading .ass procedurals during rendering. (#2795)
- **Enhanced backtrace dumps:** Backtrace dumps for crashes in debug builds are now cleaner and show more helpful information, along with colored output and time of the crash. If available, it will also report the pixel location where the crash occurred. (#2716, #2717, #2831, #2832)
- **Added console colors to log messages in Windows:** The log output sent to the console will now use colors to highlight warnings in yellow and errors in red. This feature was already available in Linux/OSX but was missing in Windows. (#2830)

#### API additions

#### Incompatible changes

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2602	Cleanup Linux memory reporting	arnold	thiago	major	3.3	5 months
#2797	Running out of memory with high ray depth	arnold	thiago	major	3.3	6 weeks
#2798	Crash writing bump2d to .ass with a value set to the "shader" parameter	arnold	angel	major	3.3	6 weeks
#2800	Bug in Python API functions returning a pointer	arnold	angel	major	3.3	5 weeks
#2812	OIIO failed assertion with accumulated error messages	arnold	ramon	major	3.3	5 weeks
#2818	memory leak when closing plugin file handle on linux/os x	arnold	thiago	major	3.3	4 weeks
#2820	NANs in Ward-Duer BRDF with subdiv_smooth_derivs enabled	arnold	alan	major	3.3	4 weeks
#2821	rare crash in Ward-Duer BRDF when used with volume_scattering	arnold	alan	major	3.3	4 weeks
#2718	Long multi-paths are cut out with "kick -l"	arnold	oscar	minor	3.3	3 months
#2803	"included" .ass files should be taken into account in ".ass parsing" time	arnold	oscar	minor	3.3	5 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2844	Wrong gamma correction on binary encoded arrays	4.0.5.1	arnold	angel	major	4.0.6
#2854	UV-based displacement combined with deformation motion blur broken	4.0.5.1	arnold	mike	major	4.0.6

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2838	popping artifacts in multi-threaded displacement	framestore, 4.0.5.2	arnold	oscar	major	4.0.6

## 4.0.5.1

### Milestone 4.0.5

#### Enhancements

- **Multi-threaded ray accel build for deformation motion blur:** Similar to the 4.0.3 release (#2653) that added support for building non-motion blurred objects in parallel, we now allow for multiple render threads to build a motion blurred object's ray accel in parallel. On a 12 core machine this lets us build the accel structure for a hair object 9.5x faster. (#2755)
- **Miscellaneous ray accel and subdivision optimizations:** The ray accel structure for deformation motion blurred objects now uses 7% less memory. Ray accel structure builds, especially when using many threads, will now be a percent or two faster. Triangulation is now roughly 2x faster; as this is also used at the end of the Catmull-Clark subdivision process (which generates quads that need to be triangulated), subdivision will also be slightly faster, around 10-15% faster overall. (#2816, #2811, #2817)
- **Multi-threaded .ass loading:** The .ass parsing code has been rewritten and made thread-safe, which allows multiple .ass-based procedurals to be processed in parallel, as it already happened with DSO and funcptr procedurals. If needed, parallel loading can be disabled with options.enable\_threaded\_procedurals. (#2113)
- **Oriented curves:** The curves node now has an additional mode oriented that allows the specification of orientations via the orientations vector array. It behaves similar to the ribbon mode, except curves can face away from the camera or incoming ray according to the orientations; curves also get thinner when they face away. Uses include grass, wider ribbons, or even leaves. Orientations are specified per-point, the same number as there are control points. (#797, #2824)
- **Accurate memory reporting on Linux:** The memory usage on Linux is now correctly reported. The older code was reporting virtual memory which is significantly larger than what is really being used. The new code reports both the memory being used on physical RAM (resident memory) as well as the memory in swap. This enhanced reporting has some overhead on older Linux distros, so scenes with thousands of objects and/or log messages might take a few hundred milliseconds longer. On at least Linux kernel 2.6.34 or RHEL6/CentOS6 and newer there is no overhead. (#2602)
- **Checkpointing renders:** EXR and TIFF drivers now support render checkpointing (or "append mode"). By setting the corresponding output driver's .append attribute to true Arnold will preserve previously rendered tiles and only work on the missing ones, appending them to the output files. If no image is present the render will proceed as normal creating a new image. If image specifications do not match the render will be aborted. (#2417)
- **Deferred .ass parsing time in log stats:** The .ass parsing time reported in the log statistics is now split into two categories: parsing when loading the main scene prior to rendering, and parsing of deferred-loading .ass procedurals during rendering. (#2795)
- **Enhanced backtrace dumps:** Backtrace dumps for crashes in debug builds are now cleaner and show more helpful information, along with colored output and time of the crash. If available, it will also report the pixel location where the crash occurred. (#2716, #2717, #2831, #2832)
- **Added console colors to log messages in Windows:** The log output sent to the console will now use colors to highlight warnings in yellow and errors in red. This feature was already available in Linux/OSX but was missing in Windows. (#2830)

#### API additions

#### Incompatible changes

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2602	Cleanup Linux memory reporting	arnold	thiago	major	3.3	5 months
#2797	Running out of memory with high ray depth	arnold	thiago	major	3.3	5 weeks
#2798	Crash writing bump2d to .ass with a value set to the "shader" parameter	arnold	angel	major	3.3	5 weeks
#2800	Bug in Python API functions returning a pointer	arnold	angel	major	3.3	4 weeks
#2812	OIIO failed assertion with accumulated error messages	arnold	ramon	major	3.3	4 weeks
#2818	memory leak when closing plugin file handle on linux/os x	arnold	thiago	major	3.3	3 weeks
#2820	NANs in Ward-Duer BRDF with subdiv_smooth_derivs enabled	arnold	alan	major	3.3	3 weeks
#2821	rare crash in Ward-Duer BRDF when used with volume_scattering	arnold	alan	major	3.3	3 weeks
#2718	Long multi-paths are cut out with "kick -l"	arnold	oscar	minor	3.3	3 months
#2803	"included" .ass files should be taken into account in ".ass parsing" time	arnold	oscar	minor	3.3	4 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2844	Wrong gamma correction on binary encoded arrays	4.0.5.1	arnold	angel	major	4.0.6
#2854	UV-based displacement combined with deformation motion blur broken	4.0.5.1	arnold	mike	major	4.0.6

## 4.0.5.0

### Milestone 4.0.5

#### Enhancements

- **Multi-threaded ray accel build for deformation motion blur:** Similar to the 4.0.3 release (#2653) that added support for building non-motion blurred objects in parallel, we now allow for multiple render threads to build a motion blurred object's ray accel in parallel. On a 12 core machine this lets us build the accel structure for a hair object 9.5x faster. (#2755)
- **Miscellaneous ray accel and subdivision optimizations:** The ray accel structure for deformation motion blurred objects now uses 7% less memory. Ray accel structure builds, especially when using many threads, will now be a percent or two faster. Triangulation is now roughly 2x faster; as this is also used at the end of the Catmull-Clark subdivision process (which generates quads that need to be triangulated), subdivision will also be slightly faster, around 10-15% faster overall. (#2816, #2811, #2817)
- **Multi-threaded .ass loading:** The .ass parsing code has been rewritten and made thread-safe, which allows multiple .ass-based procedurals to be processed in parallel, as it already happened with DSO and funcptr procedurals. If needed, parallel loading can be disabled with options.enable\_threaded\_procedurals. (#2113)
- **Oriented curves:** The curves node now has an additional mode oriented that allows the specification of orientations via the orientations vector array. It behaves similar to the ribbon mode, except curves can face away from the camera or incoming ray according to the orientations; curves also get thinner when they face away. Uses include grass, wider ribbons, or even leaves. Orientations are specified per-point, the same number as there are control points. (#797, #2824)
- **Accurate memory reporting on Linux:** The memory usage on Linux is now correctly reported. The older code was reporting virtual memory which is significantly larger than what is really being used. The new code reports both the memory being used on physical RAM (resident memory) as well as the memory in swap. This enhanced reporting has some overhead on older Linux distros, so scenes with thousands of objects and/or log messages might take a few hundred milliseconds longer. On at least Linux kernel 2.6.34 or RHEL6/CentOS6 and newer there is no overhead. (#2602)
- **Checkpointing renders:** EXR and TIFF drivers now support render checkpointing (or "append mode"). By setting the corresponding output driver's .append attribute to true Arnold will preserve previously rendered tiles and only work on the missing ones, appending them to the output files. If no image is present the render will proceed as normal creating a new image. If image specifications do not match the render will be aborted. (#2417)
- **Deferred .ass parsing time in log stats:** The .ass parsing time reported in the log statistics is now split into two categories: parsing when loading the main scene prior to rendering, and parsing of deferred-loading .ass procedurals during rendering. (#2795)
- **Enhanced backtrace dumps:** Backtrace dumps for crashes in debug builds are now cleaner and show more helpful information, along with colored output and time of the crash. If available, it will also report the pixel location where the crash occurred. (#2716, #2717, #2831, #2832)
- **Added console colors to log messages in Windows:** The log output sent to the console will now use colors to highlight warnings in yellow and errors in red. This feature was already available in Linux/OSX but was missing in Windows. (#2830)

#### API additions

#### Incompatible changes

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2602	Cleanup Linux memory reporting	arnold	thiago	major	3.3	5 months
#2797	Running out of memory with high ray depth	arnold	thiago	major	3.3	3 weeks
#2798	Crash writing bump2d to .ass with a value set to the "shader" parameter	arnold	angel	major	3.3	3 weeks
#2800	Bug in Python API functions returning a pointer	arnold	angel	major	3.3	2 weeks
#2812	OIIO failed assertion with accumulated error messages	arnold	ramon	major	3.3	2 weeks
#2818	memory leak when closing plugin file handle on linux/os x	arnold	thiago	major	3.3	10 days
#2820	NaNs in Ward-Duer BRDF with subdiv_smooth_derivs enabled	arnold	alan	major	3.3	9 days
#2821	rare crash in Ward-Duer BRDF when used with volume_scattering	arnold	alan	major	3.3	9 days
#2718	Long multi-paths are cut out with "kick -l"	arnold	oscar	minor	3.3	2 months
#2803	"included" .ass files should be taken into account in ".ass parsing" time	arnold	oscar	minor	3.3	2 weeks

## 4.0.4.0

### Milestone 4.0.4

#### Enhancements

- **EXR zips compression support:** The compression parameter in the driver\_exr node now accepts a zips (single-scanline ZIP) value. zips is a more efficient compression format for Nuke. (#2770)
- **Optimized scenes with many nodes:** A few optimizations have made handling of scenes with millions of nodes much faster. In particular, node creation (specially instances), node destruction, and iteration over the nodes in general has been optimized. The memory footprint has also been reduced for these scenes. (#2725, #2778)
- **Reuse main thread for rendering:** The main Arnold thread is now also used for rendering buckets, so instead of firing N+1 threads, we now fire N threads. In some specific situations, such as with older versions of Linux, this can result in consistently faster threading scalability. (#2761)
- **Added spherical\_camera node:** A new camera type spherical\_camera has been added to help create environment maps in the lat-long format easily. To get the full spherical range, the camera's screen window must be set to [-1,-1] to [1,1]. Note that the same mapping could be achieved in the cyl\_camera with careful tweaking of the horizontal\_fov, vertical\_fov and projective parameters, but this was too convoluted to be practical. (#2785)
- **Support for multiple license servers:** We now allow for the license check to span multiple license servers. For this we added support for new environment variables so, in addition to the existing Arnold-specific ARNOLD\_LICENSE\_HOST and ARNOLD\_LICENSE\_PORT, we now also recognize the RLM-standard variables RLM\_LICENSE and solidangle\_LICENSE. Licenses will be searched for in the following order: (#2691)
  - First, from the contents of solidangle\_LICENSE, which can contain a list of multiple servers/ports in the following format:

```
server1:server2:server3:...:serverN    <--- linux/osx
server1;server2;server3;...;serverN    <--- windows
```

where serverN can be port@host or host@port (the port is optional and defaults to 5053).
  - Second, from the contents of RLM\_LICENSE, with the same format as above.
  - Finally, from the contents of ARNOLD\_LICENSE\_HOST/ARNOLD\_LICENSE\_PORT, which together define a single license server target. (it wasn't possible to specify multiple servers in these variables and it's still not possible - that is why we have added support for the above variables)
- **Upgraded OIIO to 0.10.7:** There have been several bugfixes and performance improvements since the last OIIO version we shipped with Arnold (which was 0.10.4) including: (#2766)
  - Threading performance improvement in the texture system, which should help when there are many texture file opens per second.
  - Fix for degenerate derivatives that could corrupt the filter footprint calculations in anisotropic filtering, resulting in an infinitely long major axis.
  - More efficient subpixel filtering for very narrow anisotropic footprints when on the highest-res MIP level.

#### API additions

- **AiLicenseInfo():** This API function has been added as a replacement for the older AiLicenseGetInfo() as we now get license information from all available license servers at the same time. (#2691)

```
AI_API int AiLicenseInfo(std::vector<AtLicenseInfo>& licenses);
```

#### Incompatible changes

- **Color scheme in utility shader has changed:** This affects color modes obj, prims and floatgrid. The hashing of values to RGB colors has been modified to avoid producing black or very dark colors. In particular, the object and primitive with index 0 would previously always map to black. (#2773)
- **Deprecated AiLicenseGetInfo():** This older function has changed behaviour in that it will now list license information for all license servers found, including those in the new environment variables mentioned above, in addition to the license server set via the host and port arguments. These function arguments are therefore useless and that's why this API has been deprecated, and a new AiLicenseInfo() API that has no extra arguments has been added instead. (#2691)

```
AI_API int AiLicenseGetInfo(const char* host, unsigned int port, std::vector<AtLicenseInfo>& licenses);
```

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2767	parallel BVH build hangs with degenerate triangles	arnold	thiago	major	3.3	11 days

## 4.0.3.0

### Milestone 4.0.3

#### Enhancements

- **Multi-threaded ray accel build:** Multiple render threads can now build an object's ray accel in parallel, whereas previously only the first thread that touched the object would work on it and the rest of the threads would be forced to wait for it to finish. In machines with many cores, this produces big speedups, specially in scenes with a single multi-million polygon hero object that fills most of the image. We see close to ideal scaling (11.8x faster on a hyper threaded machine with 12 physical cores) for meshes with millions of polygons, and good scaling for other types of objects (8x faster for points). We have not yet implemented this for motion blurred ray accels. (#2653)
- **Added options.enable\_fast\_lights:** The optimizations for scenes with many lights that were introduced in 4.0.2 can now be switched off with this global render option. This is useful when benchmarking and debugging but we don't recommend switching it off in production. (#2731)
- **Upgraded RLM to 9.3BL2:** We have upgraded the license server and the external library controlling the licensing subsystem from version 9.2BL2 to 9.3BL2. The new RLM license server increased the maximum number of file descriptors from 1024 to 8192, greatly alleviating problems in environments with a high number of license requests, e.g. farms with near a thousand render nodes. (#2741)
- **Pre-configured RLM server timeout:** We now distribute the solidangle.opt options file for the ISV solidangle, along with the rlm server and the solidangle.set settings file. This file (solidangle.opt) pre-configures the TIMEOUT feature in the server with a value of 120 seconds. If necessary, users can still manually change this value as explained in the Licensing wiki. (#2742)
- **Show license environment with kick -licensecheck:** In addition to information like the product version and license count, the -licensecheck option now prints the contents of the environment variables related to licensing that Arnold listens to. (#2753)
- **Example Linux script for starting/stopping RLM as a service:** We now distribute an example script rlmd for starting and stopping the RLM server as a service in Linux. (#2743)

#### API additions

- **fast\_expf():** Added a significantly faster alternative to the standard expf() function that is within 300 ulp of the actual value (error is about 3e-5 for commonly used values) and is guaranteed correct at exp(0) and exp(1). This can be helpful for shaders where the heavy use of expf() becomes a performance bottleneck, such as scattering shaders. (#2700)

#### Incompatible changes

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2451	Artifact in lighting from a 'cylinder_light' with MIS enabled	arnold	alan	major	3.3	5 months
#2728	crash when parsing .obj files with no texture indices	arnold	angel	major	3.3	10 days
#2729	long pause when measuring string pool size	arnold	angel	major	3.3	10 days
#2730	root AtAggrList node is not being marked unbounded	arnold	thiago	major	3.3	10 days
#2744	wrong gamma correction in shader RGB arrays when written to .ass	arnold	angel	major	3.3	6 days
#2749	excessively verbose threading messages for displacement, skydome, SSS	arnold	marcos	major	3.3	3 days
#2751	Arrays can not handle more than 4GB of data	arnold	thiago	major	3.3	44 hours
#2745	Crash when using a volume_scattering shader with samples parameter set to zero	arnold	marcos	minor	3.3	6 days
#2747	remove useless optimization message about multi-key motion matrices	arnold	alan	minor	3.3	3 days
#2732	thread id not being set on shading globals passed to field shader	arnold	ramon	trivial	3.3	9 days

## 4.0.2.0

### Milestone 4.0.2

#### Enhancements

- **Multi-threaded displacement:** When two or more buckets require the result of a mesh displacement in order to continue they may now collaborate in the displacement computation to speed it up when the enable\_threaded\_displacement render option is active (*default ON*). (#2212)
- **Displacement of un-subdivided meshes:** It is no longer required to set polymesh.subdiv\_type to other than none before displacing a polygon mesh. This is useful for dense meshes that already have enough polygons to capture the detail in the displacement map. (#2212)
- **Per-face displacement shaders:** polymesh.disp\_map has been changed from an AtNode parameter to an array of AtNode's. So now, not only does it accept a NULL or a single shader as input, but also an array of displacement shaders. Please note that when it contains more than one shader, polymesh.disp\_map will share the indexing scheme provided by polymesh.shidxs and as a result must have the same number of elements as polymesh.shader. (#2669)
- **Faster abort during displacement and ray accel build:** The renderer will now more frequently poll for a render abort condition during both displacement and ray accel builds, which increases interactivity when working with big meshes and high subdiv settings. (#2709, #2710)
- **Faster scenes with many lights:** Scenes with thousands of lights cause quite long render times simply due to the time it takes to loop over the lights themselves. Some of those lights probably won't have any influence for a given shading point. We have introduced an acceleration structure to classify the lights by their volumes of influence and then loop over the important lights only. sg->lights and sg->nights no longer return all the lights in the scene, and instead they now return the (hopefully small) set of lights which can have any influence in sg->P. The speedup will depend on both the number of lights and the number of overlapping influence volumes. We have seen speedups of 1.5x in empty scenes with 4 sparse lights all the way up to 20x in empty scenes with a few hundred sparse lights. (#2542, #2543)
- **New nights color mode in utility shader:** The nights mode shows the relative number of lights considered at the shading point, which is helpful when debugging scenes with many lights. This is visualized with a "heatmap" color gradient that goes from red (all lights in the scene are considered), passing through yellow and green, to blue (very few lights are considered) and finally black (no lights considered). (#2664)
- **Auto-cropped EXR output:** We now optionally embed DataWindow information when saving EXR files. This data window, or ROI in Nuke, stores a tight bounding box around non-empty pixels in the image, which can greatly accelerate post-processing. To enable this feature, use theautocrop attribute in the driver\_exr node (*default OFF*). (#2660)
- **Removed default node name:** When creating a new node, instead of this node receiving a default name based on a combination of timestamp, node type and count, etc... we now just give the node an empty name. This results in memory savings and speedups when loading or writing .ass files. (#2649)
- **Better info messages for licensing:** We have updated the warning messages for the licensing subsystem, making them more informative. This also includes messages coming from the kick -licensecheck command. (#2638, #2657)

- No server running:  
00:00:00 5MB WARNING | [rlm] could not connect to license server on 5053@localhost
- Wrong license version:  
00:00:00 5MB WARNING | [rlm] wrong license version, found 1 license for arnold 303  
00:00:00 5MB WARNING | [rlm] please contact licensing@solidangle.com
- Expired license:  
00:00:00 5MB WARNING | [rlm] could not find any license for arnold 400, the license may be expired  
00:00:00 5MB WARNING | [rlm] please contact licensing@solidangle.com
- No license files:  
00:00:00 5MB WARNING | [rlm] could not find any license files, please check the license server
- Up-to-date license file:  
00:00:00 5MB | [rlm] checkout of arnold (version 400) OK

#### API additions

- **AiArrayInterpolateRGB/RGBA():** In Arnold 4.0.0 we introduced several AiArrayInterpolate\*() functions but we somehow forgot the RGB and RGBA versions. (#2666)

#### Incompatible changes

- **Transfer control about aborting on texture error to the calling code:** By setting a valid pointer for the return (success) value

in AiTextureAccess(), the current error message is disabled and the render process is not aborted. It is left for the client code to handle the error as appropriate. (#2671)

- **Deprecated some 2D vector functions:** The old-style AiV2Add, AiV2Sub and AiV2Scale functions have been deprecated, the operators +, -, \* should be used instead. The signature of the AiV2Lerp and AiV2Clamp functions has changed so that they return a value directly rather than through an argument; the old-style functions have been kept for now but marked as deprecated. (#2648)
- **disp\_height no longer affects displaced mesh bounding box:** Both disp\_height and disp\_padding are separate and independent now. The former sets scale for displacement, while the latter extends the bounding box. If disp\_height is set without setting disp\_padding, it could result in clipping of the displaced mesh. (#2722)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2678	arnold 4 and auto_transparency_threshold	arnold	alan	blocker	3.3	3 weeks
#2656	always_linear metadata doesn't work when writing to .ass	arnold	angel	critical	3.3	5 weeks
#2659	Weird shadowing with transformed instances of procedurals	arnold	oscar	critical	3.3	5 weeks
#2662	Halo artifacts with 'importance_sampling' in 'volume_scattering'	arnold	oscar	critical	3.3	5 weeks
#2665	Bug when writing to .ass shaders with component links	arnold	angel	critical	3.3	4 weeks
#2479	Some geometry in procedurals with SSS is not rendered	arnold	oscar	major	3.3	4 months
#2641	Threading Issue when generating SSS pointclouds	arnold	oscar	major	3.3	6 weeks
#2646	intersection precision problems in cone primitive	arnold	oscar	major	3.3	6 weeks
#2652	Handle ABS(int)	arnold	thiago	major	3.3	5 weeks
#2655	Linking to a component in an array element not working	arnold	angel	major	3.3	5 weeks
#2661	missing Doxygen docs in AiSSSEvaluateIrradiance()	arnold	thiago	major	3.3	5 weeks
#2675	missing return value from AiRender() function in Python bindings	arnold	angel	major	3.3	3 weeks
#2676	missing return value from AiASS*() functions in Python bindings	arnold	oscar	major	3.3	3 weeks
#2677	Bad composition of inverse XForm matrices in procedural networks	arnold	oscar	major	3.3	3 weeks
#2680	Memory leak on array element linking	arnold	angel	major	3.3	3 weeks
#2681	Wrong shader override in instanced procedural networks	arnold	oscar	major	3.3	3 weeks
#2682	Wrong xform matrix recovering in instanced procedural networks	arnold	oscar	major	3.3	3 weeks
#2683	missing return value from AiNodeGetLink() function in Python bindings	arnold	angel	major	3.3	3 weeks
#2685	AiFormatTime() is not thread-safe	arnold	marcos	major	3.3	3 weeks
#2698	IPR memory leak in utility node in ambocc mode	arnold	angel	major	3.3	2 weeks
#2701	Nodes with empty names cause problems when saving to .ass	arnold	angel	major	3.3	10 days
#2704	AI_LOG_SSS missing from python bindings	arnold	angel	major	3.3	8 days
#2711	Misaligned temporary SIMD variables in MSVC cause memory crashes	arnold	xo	major	3.3	4 days
#2713	Gamma is not being applied to RGB(A) array elements	arnold	angel	major	3.3	3 days
#2714	log memory stamps shift by 1 above 10Gb	arnold	marcos	major	3.3	3 days
#2715	wireframe returns black in indirect	arnold	marcos	major	3.3	3 days
#2679	P color mode in utility shader is broken	arnold	oscar	minor	3.3	3 weeks

## 4.0.1.3

### Milestone 4.0.1

#### Enhancements

- **Added enable\_threaded\_procedurals:** This global option allows users to enable or disable the parallel loading of procedural nodes. This is normally enabled by default, but it can be disabled as a convenience to avoid crashes with procedural code that is not yet thread-safe (e.g. Katana). Rather than disabling this option, we urge developers to make their code thread-safe to take full advantage of the capabilities of modern hardware. (#2637)
- **Reduced peak memory usage for deformation motion blurred ray accels:** The peak memory usage while building the ray accel data structure for deformation motion blurred objects has been significantly reduced, often by half. This can help when rendering large deformation motion blurred objects such as multi-million polygon/hair objects. Note that this memory peak is usually not reported by the Arnold statistics, but the effect of this optimization can still be noticed by faster build times when previous versions ran out of physical memory. (#2631)
- **Faster deformation motion blurred ray accels:** Objects with extreme deformation motion blur should now render faster. In one particular production scene with exploding, very fast moving particles, this resulted in a 200x speedup. Scenes with average deformation won't see much or any speedup. This does however result in about a 25% slower build time. (#2635)
- **Faster ray accels:** Fixed a bug in ray accel creation that was generating inefficient (but still valid and correct) ray accels. Render time should be several percent faster; we've seen 7% in our tests for architectural scenes. (#1526)
- **Faster auto-transparency AOV blending:** The AOV composition code has been optimized, greatly reducing the overhead incurred when the enable\_aov\_composition render option is set. Scenes with shaders that export many different AOVs yet only actually write a few to disk will see the most benefit. (#2634,#2639)
- **Faster gamma correction in handle-based texture lookups:** The inverse-gamma correction for handle-based lookups of LDR texture maps has been optimized resulting in up to 1.2x speedups in simple scenes. The older filename-based texture lookups are unaffected by this change, as are scenes that have no inverse-gamma correction, i.e. HDR maps or LDR maps with options->texture\_gamma == 1.0. (#2628)
- **Faster approximate expf():** The internal expf() function has been approximated and optimized, resulting in ~3% speedups in simple scenes which heavily use that function (gaussian filters, fog shaders, etc). (#2629)

#### API additions

- **Arithmetic operators for AtPoint2:** AtPoint2 was missing the overloaded operators (+,-,\*,=, etc...) present in AtPoint. (#2621)

```
AtPoint2 a, b;  
...  
AtPoint2 c = a + b;
```

- **rgb() accessor in AtRGBA:** It is now possible to get and set the RGB part in an AtRGBA with the rgb() accessor, and use it for common AtRGB operations. This can be useful in code that needs to mix RGB and RGBA colors: (#2633)

```
AtRGBA rgba1, rgba2;  
...  
AtColor c = rgba1.rgb() + rgba2.rgb();  
rgba2.rgb() = rgba1.rgb();
```

#### Incompatible changes

- **Deprecated legacy utility functions:** The following scalar utility functions from ai\_types.h have been deprecated: MAP01(), MAP01F(), INVERSE(), ACOSF(), AiBerpScalar(). As far as we can tell, they are unused, or in the case of ACOSF() superseded by a templated version that works on both floats and doubles. (#2622)
- **Templatized BILERP() and BIHERP():** Following all the other utilities in ai\_types.h, these two scalar utilities have been templatized so that they can work on both floats and doubles. In addition, they now return a value instead of void. The old non-returning versions have been kept but deprecated. (#2622)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#1526	SAH equation in BVH builder is wrong	arnold	thiago	major	3.0	2 years
#2620	kick -licensecheck has stopped giving license details	kick	oscar	major	3.3	6 weeks
#2622	ai_types.h is not exposed in the Doxygen html	arnold	marcos	major	3.3	6 weeks
#2632	Symbols are not correctly stripped in libai.dylib	arnold	oscar	major	3.3	5 weeks
#2642	wrong self-hit treatment in cone/cylinder	arnold	oscar	major	3.3	4 weeks
#2644	'box' count is not reported in the initial log messages	arnold	oscar	trivial	3.3	4 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2656	always_linear metadata doesn't work when writing to .ass	4.0.1.1	arnold	angel	critical	4.0.2
#2646	intersection precision problems in cone primitive	4.0.1.1	arnold	oscar	major	4.0.2
#2652	Handle ABS(int)	4.0.1.1	arnold	thiago	major	4.0.2
#2655	Linking to a component in an array element not working	4.0.1.1	arnold	angel	major	4.0.2

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2662	Halo artifacts with 'importance_sampling' in 'volume_scattering'	framestore, 4.0.1.2	arnold	oscar	critical	4.0.2
#2665	Bug when writing to .ass shaders with component links	4.0.1.2	arnold	angel	critical	4.0.2
#2641	Threading Issue when generating SSS pointclouds	framestore, 4.0.1.2	arnold	oscar	major	4.0.2
#2661	missing Doxygen docs in AiSSSEvaluateIrradiance()	4.0.1.2	arnold	thiago	major	4.0.2

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2678	arnold 4 and auto_transparency_threshold	4.0.1.3	arnold	alan	blocker	4.0.2
#2479	Some geometry in procedurals with SSS is not rendered	4.0.1.3	arnold	oscar	major	4.0.2
#2677	Bad composition of inverse XForm matrices in procedural networks	4.0.1.3, 3.3.13.3	arnold	oscar	major	4.0.2
#2680	Memory leak on array element linking	4.0.1.3	arnold	angel	major	4.0.2

## 4.0.1.2

### Milestone 4.0.1

#### Enhancements

- **Added enable\_threaded\_procedurals:** This global option allows users to enable or disable the parallel loading of procedural nodes. This is normally enabled by default, but it can be disabled as a convenience to avoid crashes with procedural code that is not yet thread-safe (e.g. Katana). Rather than disabling this option, we urge developers to make their code thread-safe to take full advantage of the capabilities of modern hardware. (#2637)
- **Reduced peak memory usage for deformation motion blurred ray accels:** The peak memory usage while building the ray accel data structure for deformation motion blurred objects has been significantly reduced, often by half. This can help when rendering large deformation motion blurred objects such as multi-million polygon/hair objects. Note that this memory peak is usually not reported by the Arnold statistics, but the effect of this optimization can still be noticed by faster build times when previous versions ran out of physical memory. (#2631)
- **Faster deformation motion blurred ray accels:** Objects with extreme deformation motion blur should now render faster. In one particular production scene with exploding, very fast moving particles, this resulted in a 200x speedup. Scenes with average deformation won't see much or any speedup. This does however result in about a 25% slower build time. (#2635)
- **Faster ray accels:** Fixed a bug in ray accel creation that was generating inefficient (but still valid and correct) ray accels. Render time should be several percent faster; we've seen 7% in our tests for architectural scenes. (#1526)
- **Faster auto-transparency AOV blending:** The AOV composition code has been optimized, greatly reducing the overhead incurred when the enable\_aov\_composition render option is set. Scenes with shaders that export many different AOVs yet only actually write a few to disk will see the most benefit. (#2634,#2639)
- **Faster gamma correction in handle-based texture lookups:** The inverse-gamma correction for handle-based lookups of LDR texture maps has been optimized resulting in up to 1.2x speedups in simple scenes. The older filename-based texture lookups are unaffected by this change, as are scenes that have no inverse-gamma correction, i.e. HDR maps or LDR maps with options->texture\_gamma == 1.0. (#2628)
- **Faster approximate expf():** The internal expf() function has been approximated and optimized, resulting in ~3% speedups in simple scenes which heavily use that function (gaussian filters, fog shaders, etc). (#2629)

#### API additions

- **Arithmetic operators for AtPoint2:** AtPoint2 was missing the overloaded operators (+,-,\*,=, etc...) present in AtPoint. (#2621)

```
AtPoint2 a, b;  
...  
AtPoint2 c = a + b;
```

- **rgb() accessor in AtRGBA:** It is now possible to get and set the RGB part in an AtRGBA with the rgb() accessor, and use it for common AtRGB operations. This can be useful in code that needs to mix RGB and RGBA colors: (#2633)

```
AtRGBA rgba1, rgba2;  
...  
AtColor c = rgba1.rgb() + rgba2.rgb();  
rgba2.rgb() = rgba1.rgb();
```

#### Incompatible changes

- **Deprecated legacy utility functions:** The following scalar utility functions from ai\_types.h have been deprecated: MAP01(), MAP01F(), INVERSE(), ACOSF(), AiBerpScalar(). As far as we can tell, they are unused, or in the case of ACOSF() superseded by a templated version that works on both floats and doubles. (#2622)
- **Templatized BILERP() and BIHERP():** Following all the other utilities in ai\_types.h, these two scalar utilities have been templatized so that they can work on both floats and doubles. In addition, they now return a value instead of void. The old non-returning versions have been kept but deprecated. (#2622)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#1526	SAH equation in BVH builder is wrong	arnold	thiago	major	3.0	2 years
#2620	kick -licensecheck has stopped giving license details	kick	oscar	major	3.3	5 weeks
#2622	ai_types.h is not exposed in the Doxygen html	arnold	marcos	major	3.3	5 weeks
#2632	Symbols are not correctly stripped in libai.dylib	arnold	oscar	major	3.3	3 weeks
#2642	wrong self-hit treatment in cone/cylinder	arnold	oscar	major	3.3	2 weeks
#2644	'box' count is not reported in the initial log messages	arnold	oscar	trivial	3.3	2 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2656	always_linear metadata doesn't work when writing to .ass	4.0.1.1	arnold	angel	critical	4.0.2
#2646	intersection precision problems in cone primitive	4.0.1.1	arnold	oscar	major	4.0.2
#2652	Handle ABS(int)	4.0.1.1	arnold	thiago	major	4.0.2
#2655	Linking to a component in an array element not working	4.0.1.1	arnold	angel	major	4.0.2

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2662	Halo artifacts with 'importance_sampling' in 'volume_scattering'	framestore, 4.0.1.2	arnold	oscar	critical	4.0.2
#2665	Bug when writing to .ass shaders with component links	4.0.1.2	arnold	angel	critical	4.0.2
#2641	Threading Issue when generating SSS pointclouds	framestore, 4.0.1.2	arnold	oscar	major	4.0.2
#2661	missing Doxygen docs in AiSSSEvaluateIrradiance()	4.0.1.2	arnold	thiago	major	4.0.2

## 4.0.1.1

### Milestone 4.0.1

#### Enhancements

- **Added enable\_threaded\_procedurals:** This global option allows users to enable or disable the parallel loading of procedural nodes. This is normally enabled by default, but it can be disabled as a convenience to avoid crashes with procedural code that is not yet thread-safe (e.g. Katana). Rather than disabling this option, we urge developers to make their code thread-safe to take full advantage of the capabilities of modern hardware. (#2637)
- **Reduced peak memory usage for deformation motion blurred ray accels:** The peak memory usage while building the ray accel data structure for deformation motion blurred objects has been significantly reduced, often by half. This can help when rendering large deformation motion blurred objects such as multi-million polygon/hair objects. Note that this memory peak is usually not reported by the Arnold statistics, but the effect of this optimization can still be noticed by faster build times when previous versions ran out of physical memory. (#2631)
- **Faster deformation motion blurred ray accels:** Objects with extreme deformation motion blur should now render faster. In one particular production scene with exploding, very fast moving particles, this resulted in a 200x speedup. Scenes with average deformation won't see much or any speedup. This does however result in about a 25% slower build time. (#2635)
- **Faster ray accels:** Fixed a bug in ray accel creation that was generating inefficient (but still valid and correct) ray accels. Render time should be several percent faster; we've seen 7% in our tests for architectural scenes. (#1526)
- **Faster auto-transparency AOV blending:** The AOV composition code has been optimized, greatly reducing the overhead incurred when the enable\_aov\_composition render option is set. Scenes with shaders that export many different AOVs yet only actually write a few to disk will see the most benefit. (#2634,#2639)
- **Faster gamma correction in handle-based texture lookups:** The inverse-gamma correction for handle-based lookups of LDR texture maps has been optimized resulting in up to 1.2x speedups in simple scenes. The older filename-based texture lookups are unaffected by this change, as are scenes that have no inverse-gamma correction, i.e. HDR maps or LDR maps with options->texture\_gamma == 1.0. (#2628)
- **Faster approximate expf():** The internal expf() function has been approximated and optimized, resulting in ~3% speedups in simple scenes which heavily use that function (gaussian filters, fog shaders, etc). (#2629)

#### API additions

- **Arithmetic operators for AtPoint2:** AtPoint2 was missing the overloaded operators (+,-,\*,=, etc...) present in AtPoint. (#2621)

```
AtPoint2 a, b;  
...  
AtPoint2 c = a + b;
```

- **rgb() accessor in AtRGBA:** It is now possible to get and set the RGB part in an AtRGBA with the rgb() accessor, and use it for common AtRGB operations. This can be useful in code that needs to mix RGB and RGBA colors: (#2633)

```
AtRGBA rgba1, rgba2;  
...  
AtColor c = rgba1.rgb() + rgba2.rgb();  
rgba2.rgb() = rgba1.rgb();
```

#### Incompatible changes

- **Deprecated legacy utility functions:** The following scalar utility functions from ai\_types.h have been deprecated: MAP01(), MAP01F(), INVERSE(), ACOSF(), AiBerpScalar(). As far as we can tell, they are unused, or in the case of ACOSF() superseded by a templated version that works on both floats and doubles. (#2622)
- **Templatized BILERP() and BIHERP():** Following all the other utilities in ai\_types.h, these two scalar utilities have been templatized so that they can work on both floats and doubles. In addition, they now return a value instead of void. The old non-returning versions have been kept but deprecated. (#2622)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#1526	SAH equation in BVH builder is wrong	arnold	thiago	major	3.0	2 years
#2620	kick -licensecheck has stopped giving license details	kick	oscar	major	3.3	4 weeks
#2622	ai_types.h is not exposed in the Doxygen html	arnold	marcos	major	3.3	3 weeks
#2632	Symbols are not correctly stripped in libai.dylib	arnold	oscar	major	3.3	2 weeks
#2642	wrong self-hit treatment in cone/cylinder	arnold	oscar	major	3.3	7 days
#2644	'box' count is not reported in the initial log messages	arnold	oscar	trivial	3.3	7 days

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2656	always_linear metadata doesn't work when writing to .ass	4.0.1.1	arnold	angel	critical	4.0.2
#2646	intersection precision problems in cone primitive	4.0.1.1	arnold	oscar	major	4.0.2
#2652	Handle ABS(int)	4.0.1.1	arnold	thiago	major	4.0.2
#2655	Linking to a component in an array element not working	4.0.1.1	arnold	angel	major	4.0.2

## 4.0.1.0

### Milestone 4.0.1

#### Enhancements

- **Added enable\_threaded\_procedurals:** This global option allows users to enable or disable the parallel loading of procedural nodes. This is normally enabled by default, but it can be disabled as a convenience to avoid crashes with procedural code that is not yet thread-safe (e.g. Katana). Rather than disabling this option, we urge developers to make their code thread-safe to take full advantage of the capabilities of modern hardware. (#2637)
- **Reduced peak memory usage for deformation motion blurred ray accels:** The peak memory usage while building the ray accel data structure for deformation motion blurred objects has been significantly reduced, often by half. This can help when rendering large deformation motion blurred objects such as multi-million polygon/hair objects. Note that this memory peak is usually not reported by the Arnold statistics, but the effect of this optimization can still be noticed by faster build times when previous versions ran out of physical memory. (#2631)
- **Faster deformation motion blurred ray accels:** Objects with extreme deformation motion blur should now render faster. In one particular production scene with exploding, very fast moving particles, this resulted in a 200x speedup. Scenes with average deformation won't see much or any speedup. This does however result in about a 25% slower build time. (#2635)
- **Faster ray accels:** Fixed a bug in ray accel creation that was generating inefficient (but still valid and correct) ray accels. Render time should be several percent faster; we've seen 7% in our tests for architectural scenes. (#1526)
- **Faster auto-transparency AOV blending:** The AOV composition code has been optimized, greatly reducing the overhead incurred when the enable\_aov\_composition render option is set. Scenes with shaders that export many different AOVs yet only actually write a few to disk will see the most benefit. (#2634,#2639)
- **Faster gamma correction in handle-based texture lookups:** The inverse-gamma correction for handle-based lookups of LDR texture maps has been optimized resulting in up to 1.2x speedups in simple scenes. The older filename-based texture lookups are unaffected by this change, as are scenes that have no inverse-gamma correction, i.e. HDR maps or LDR maps with options->texture\_gamma == 1.0. (#2628)
- **Faster approximate expf():** The internal expf() function has been approximated and optimized, resulting in ~3% speedups in simple scenes which heavily use that function (gaussian filters, fog shaders, etc). (#2629)

#### API additions

- **Arithmetic operators for AtPoint2:** AtPoint2 was missing the overloaded operators (+,-,\*,=, etc...) present in AtPoint. (#2621)

```
AtPoint2 a, b;  
...  
AtPoint2 c = a + b;
```

- **rgb() accessor in AtRGBA:** It is now possible to get and set the RGB part in an AtRGBA with the rgb() accessor, and use it for common AtRGB operations. This can be useful in code that needs to mix RGB and RGBA colors: (#2633)

```
AtRGBA rgba1, rgba2;  
...  
AtColor c = rgba1.rgb() + rgba2.rgb();  
rgba2.rgb() = rgba1.rgb();
```

#### Incompatible changes

- **Deprecated legacy utility functions:** The following scalar utility functions from ai\_types.h have been deprecated: MAP01(), MAP01F(), INVERSE(), ACOSF(), AiBerpScalar(). As far as we can tell, they are unused, or in the case of ACOSF() superseded by a templated version that works on both floats and doubles. (#2622)
- **Templatized BILERP() and BIHERP():** Following all the other utilities in ai\_types.h, these two scalar utilities have been templatized so that they can work on both floats and doubles. In addition, they now return a value instead of void. The old non-returning versions have been kept but deprecated. (#2622)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#1526	SAH equation in BVH builder is wrong	arnold	thiago	major	3.0	2 years
#2620	kick -licensecheck has stopped giving license details	kick	oscar	major	3.3	3 weeks
#2622	ai_types.h is not exposed in the Doxygen html	arnold	marcos	major	3.3	3 weeks
#2632	Symbols are not correctly stripped in libai.dylib	arnold	oscar	major	3.3	9 days
#2642	wrong self-hit treatment in cone/cylinder	arnold	oscar	major	3.3	26 hours
#2644	'box' count is not reported in the initial log messages	arnold	oscar	trivial	3.3	25 hours

# 4.0.0

## Milestone 4.0

### Enhancements

- **Provide displacement derivatives:** Estimates for the dPdx, dPdy, dudx, dudy, dvdx and dvdy shader globals are now being provided to the displacement shader for each vertex where it is evaluated. With this information the displacement shader now performs filtered texture map lookups, potentially reducing the amount of texture file I/O, sometimes by 10-20x depending on the size of the tessellated polygons: the bigger the polygons, the less texture will be loaded. This enhancement can change the rendered image but if so desired it can be toggled via the newenable\_displacement\_derivs render option. (#1313)
- **Smooth tangents for subdivision surfaces:** Subdivision meshes are now capable of generating "limit" dPdu & dPdv derivatives which allow for anisotropic shaders to have a smoother appearance. Previously, these derivatives were constant per triangle, which resulted in a faceted appearance. The generation of smooth derivatives can be enabled per mesh via the new subdiv\_smooth\_derivs parameter. Please note that this capability requires storing roughly 100 extra bytes of data per vertex per keyframe so we are disabling it by default. (trac#2339)
- **Multi-threaded DSO procedural loading:** DSO-based procedural nodes can now be processed in parallel, meaning that different threads can work on expanding different procedurals. Previously, there was a global lock around procedural expansion which only allowed one procedural to be expanded at a time. The new, multi-threaded code can provide significant speedups in scenes whose load time is dominated by many CPU-bound procedurals. Note that this does not yet apply to .ass-based procedurals/standins, which we plan to multi-thread in a future release. (#2108)
- **Automatic AOV blending through semi-transparent surfaces:** Via the enable\_aov\_composition render option, it is now possible for AI\_TYPE\_RGB AOVs registered with the AI\_AOV\_BLEND\_OPACITY blending mode to automatically compose through semi-opaque surfaces rendered using auto-transparency. Due to the potentially drastic changes to AOVs that this change can cause as well as the associated performance hit, AOV composition is turned off by default. Performance will be improved in a future release. (#2455)
- **Export composable AOVs from standard, lambert and hair shaders:** Taking advantage of the new AOV composition functionality, AOVs with opacity-blending capability have been added to these built-in shaders. In particular, the hair shader now has direct\_diffuse,direct\_specular and indirect\_diffuse AOVs. (#2455, #2606, #2610)
- **Reduced peak memory usage for ray accels:** The peak memory usage while building the ray tracing acceleration data structures has been significantly reduced, often by half. This can help when rendering large objects such as multi-million polygon/hair/particle objects. Note that this peak is usually not reported by the Arnold statistics, but it can still be noticed by faster build times when previous versions ran out of physical memory. This has yet to be applied to deformation blur. (#1547, #2523)
- **Reduced memory usage for importance tables:** The memory used for importance tables in the skydome\_light and in texture-mapped quad\_light nodes has been reduced to around 33% of the original by using RGBE encoding. This can have an effect in scenes with many textured area lights. (#2534)
- **Reduced memory usage for points primitive:** The disk and sphere modes of the points primitive now always use 33% less memory. If the radii are all the same, they use 50% less memory. (trac#2301)
- **Faster points primitive in quad mode:** The quad mode of the points primitive now renders about 15% faster thanks to various ray intersection optimizations. (#2495, #2607)
- **Faster motion blurred matrices:** The underlying matrix math has been vectorized, resulting in a few percent faster rendering time for scenes which have objects with motion blurred matrices. (#2570)
- **Faster trigonometric/trascendental math:** The trigonometric/trascendental math functions used inside the renderer (sin, cos, atan, exp etc) have been replaced with either fast, approximate versions, or with 4-way parallel versions. This results in 1.1x - 1.5x speedups for simple scenes. The speedups won't be as big for complex production scenes. (#2495, #2556, #2557, #2588)
- **Faster pixel filters:** The Gaussian, Blackman-Harris, Cook and Sinc filters have been optimized for speed. This can result in ~2x speedups in simple scenes with lots of empty pixels or when using a high number of AOVs and AA samples. (#2553, #2558, #2560, #2561, #2564)
- **Faster texture lookups in image node:** The texture lookups in the image shader have been optimized by using the handle-based lookup API introduced in this release. This can provide significant speedups when rendering textures with long filenames with many threads. For benchmarking purposes, the lookups can be switched to use the old, less efficient lookups by disabling the new parameter cache\_texture\_handles. (#1546)
- **base85 encoding of large float arrays:** The default encoding for float arrays in .ass files has been changed to base85, which is more efficient in both file size and read/write time over the previous base64 encoding. Older scenes written with base64 encoding or with no binary encoding at all can still be read normally. (#2444)
- **Energy preserving SSS:** The normalization factor for the cubic BSSRDF in AiSSSPointCloudLookup() was incorrect by a factor of  $4/(10/\pi)$ , or ~1.2566. We have now corrected this bug, which makes the SSS effect a bit darker but truly energy conserving. This can be verified by rendering a 100% white SSS sphere illuminated by a flat white sky, which will now be invisible as expected. Users who upgrade to this release in the middle of a show, and find that the SSS component in their materials is noticeably darker, should simply multiply Ksss by the constant 1.2566. If your show uses custom shaders, this can be easily fixed in the shader code instead of fixing all the materials: (#2535)

```
AtColor sss = AiSSSPointCloudLookupCubic() * 1.2566;
```
- **Added reference\_time:** The old sss\_time option has been renamed to reference\_time so that it can be used for two different things. On one hand, the SSS pointclouds will be computed at precisely this time (which is what sss\_time was used for). On the other hand, when disabling motion blur with the ignore\_motion\_blur option, the time specified by reference\_time will be used instead of the

shutter open time. This fixes the problem where a scene was setup with "centered" motion blur (for example: shutter=[-0.25,+0.25]) and enabling ignore\_motion\_blur would produce a scene rendered at time=-0.25 instead of time=0, which had an unexpected impact on motion vector passes. The default value of 0 will be what the user wants in 99% of cases. (#2447)

- **Implicit AOV type conversion from POINT2 to POINT/VECTOR:** A POINT2 AOV can now be stored in an image of POINT or VECTOR type. The Z component will be set to zero. (#2448)
- **Removed various unused options:** The unbounded parameter in geometry nodes, the packet\_\* global options, and the solid\_angle parameter of the quad\_light node have all been removed. These were legacy or experimental parameters that offered absolutely no value to the user and were taking unnecessary storage as well as causing confusion. (#2529, #2562, #2567)
- **Ignore options in .ass procedurals:** To avoid non-deterministic results in deferred loading of .ass files via procedural nodes, any options nodes that may have been inadvertently left inside .ass files intended for deferred loading (e.g. standins) are now filtered out and ignored. (#2519)
- **Log file enhancements:**
  - modern CPUs that were reported as "GenuineIntel" or even "Unknown Intel" are now properly reported thanks to improved CPU identification code (#2545)
  - .ass parsing warning/error messages have now the form "[ass] line <n>: <error string>" (#2483)
  - .ass line numbers are now reported more accurately (they used to be off by +-1 line) (#2486)
  - orphan node blocks in .ass files are now explicitly reported (#2472)
  - read/written bytes/nodes when parsing .ass files (#2488)
  - better warning messages when parsing unknown parameters (#2487)
  - added new warning for misplaced parameter values (#2487)
  - the active camera is now reported (#2498)
  - the version of the RLM licensing library is now reported (#2475)
- **Upgraded RLM to 9.2BL2:** We have upgraded the external library controlling the licensing subsystem from version 7.0BL4 to 9.2BL2, increasing the stability and opening the door to future enhancements. (#2411)
- **Upgraded OIIO to 0.10.4:** There have been numerous bugfixes and performance improvements since the last OIIO version we shipped with Arnold (which was 0.10.0) including: (trac#2374)
  - Improved robustness of texture lookups with very small derivatives. In some cases, this leads to a 2-3x speedup.
  - Various threading performance improvements, including fixing a dramatic slowdown when autotile is enabled.
  - Various precision-related bug fixes for PNG, ICO and Targa files.
  - maketx: fixes to sinc, blackman-harris filters.
  - maketx: --hash is deprecated, the SHA-1 hash is always computed; the hash takes into account upstream image changes, such as resizing.
  - maketx: the --filter command line argument only takes the filter name, the width is now automatically computed.
- **No need to set Arnold path for Python bindings:** Users of the Python bindings are no longer required to manually set the {DYLD|LD}\_LIBRARY\_PATH environment variable, as the Arnold path is now determined automatically. (#2578)

## API additions

- **Alternate, faster texture mapping API:** In addition to the existing filename-based texture lookup, we have added a handle-based texture lookup. This new API has much better threading scalability in particular when hyper-threading is enabled. We have seen ~1.2x speedups in texture/autobump heavy scenes, but in simple scenes with very long filenames the speedup was closer to 2x. The texture handle is typically created in the shader's node\_initialize (or better, in node\_update if the texture needs to change in IPR) and released in node\_finish. The built-in image shader has been modified to use texture handles by default. (#1546)

```
struct AtTextureHandle;

AI_API AtTextureHandle* AiTextureHandleCreate(const char* filename);
AI_API AtRGBA           AiTextureHandleAccess(const AtShaderGlobals* sg, AtTextureHandle* handle,
const AtTextureParams* params, bool* success = NULL);
AI_API void              AiTextureHandleDestroy(AtTextureHandle* handle);
```

- **AiTextureAccess() returns a success value:** An optional argument has been added to the end of the argument list which allows client applications to check for errors during texture lookups. This change is backwards-compatible. Existing client code will continue to compile and operate with no behavioural changes. The new signature is: (#2461)

```
AI_API AtRGBA AiTextureAccess(const AtShaderGlobals* sg, const char* filename, const
AtTextureParams* params, bool* success = NULL);
```

- **Added filter width controls to AtTextureParams:** The new parameters width\_s and width\_t in the AtTextureParams struct can be used to control the filter width, or blurriness, of the texture lookups. The default value is 1.0, which means no additional blurring. This removes the need for the temporary AiTextureAccessResized() symbol introduced earlier. (#2458)

```
float   width_s;    /**< amount of widening of look-ups on the S axis */
float   width_t;    /**< amount of widening of look-ups on the T axis */
```

- **Added missing AiUDataGetMatrix():** This API was missing for no good reason. The matrix type is now fully supported in constant, uniform and varying user data. (#2464, #2473, #2528, #2530)

- **AiNodeEntryInstall() / AiNodeEntryUninstall():** These functions replace the previously misnamed AiNodeInstall() / AiNodeUninstall(), which have been moved to the ai\_deprecated.h header file. (#2068)

- **Added new APIs to access AtNode fields that were previously public:** Because AtNode is now an opaque type, we are

introducing new API functions to access the missing information: (#2057)

```
AI_API const AtNodeEntry* AiNodeGetNodeEntry(const AtNode* node);
AI_API AtParamValue* AiNodeGetParams(const AtNode* node);
AI_API void* AiNodeGetLocalData(const AtNode* node);
AI_API void AiNodeSetLocalData(AtNode* node, void* data);
```

- **Added node filtering in AiASSLoad()**: The AiASSLoad() function can now load specific types of nodes, just like in AiASSWrite(). Also, default values have been added for some parameters so that client code that uses the old function signatures still compiles. The new signatures are: (#1926)

```
AI_API int AiASSWrite(const char* filename, int mask = AI_NODE_ALL, bool open_procs = FALSE);
AI_API int AiASSLoad (const char* filename, int mask = AI_NODE_ALL);
```

- **Added API for registration and iteration of AOVs**: Custom AOVs can now be registered via the AiAOVRegister() function: (#1947)

```
AI_API bool AiAOVRegister(const char* name, int type, int blend_mode = AI_AOV_BLEND_NONE);
```

When registered, these AOVs get added to the global list of AOVs, which already contains the list of built-in AOVs. This list can be iterated with the following API:

```
AI_API AtAOVIterator* AiUniverseGetAOVIterator();
AI_API void AiaAOVIteratorDestroy(AtAOVIterator* iter);
AI_API const AtAOVEntry* AiaAOVIteratorGetNext(AtAOVIterator* iter);
AI_API bool AiaAOVIteratorFinished(const AtAOVIterator* iter);
```

- **Added new shader linking API functions**: New functions have been added, and some existing ones modified, to provide support for separate component linking and array element linking. (#2052, #2053)

```
AI_API bool AiNodeLink(AtNode* src, const char* input, AtNode* target);
AI_API bool AiNodeLinkOutput(AtNode* src, const char* output, AtNode* target, const char* input);
AI_API bool AiNodeUnlink(AtNode* node, const char* input);
AI_API bool AiNodeIsLinked(const AtNode* node, const char* input);
AI_API AtNode* AiNodeGetLink(const AtNode* node, const char* input, int* comp = NULL);
```

- **Added Gaussian BSSRDF support**: Besides the already existing SSS cubic diffusion profile, a new gaussian diffusion profile is now supported. This allows shader writers to experiment with different strategies for mixing gaussians. (#2532)

```
AI_API AtColor AiSSSPointCloudLookupGaussian(const AtShaderGlobals* sg, const AtColor& variance);
```

- **Added point cloud sample API**: Methods to iterate over the SSS pointcloud samples generated by Arnold have been provided. (#2208)

```
struct AtPointCloudSample {
    AtPoint pref_position;
    AtPoint world_position;
    AtVector normal;
    AtUInt32 face;
    AtPoint2 uv;
    float area;
};

struct AtPointCloudIterator;
AI_API AtUInt32 AiPointCloudGetSampleCount(const AtShaderGlobals* sg);
AI_API AtPointCloudIterator* AiPointCloudIteratorCreate(const AtShaderGlobals* sg);
AI_API void AiPointCloudIteratorDestroy(AtPointCloudIterator* iter);
AI_API AtPointCloudSample AiPointCloudIteratorGetNext(AtPointCloudIterator* iter);
AI_API bool AiPointCloudIteratorFinished(const AtPointCloudIterator* iter);
```

- **Added method to evaluate irradiance at SSS point samples**: A new function has been added to be able to measure the irradiance at the point samples provided by the new point cloud sample API. (trac#2350)

```
AI_API AtRGB AiSSSEvaluateIrradiance(const AtShaderGlobals* sg, float u, float v, AtUInt32 face,
const AtPoint* p, AtUInt32 index);
```

- **Added default render mode in AiRender()**: The mode argument in the AiRender() API call now has a default value of AI\_RENDER\_MODE\_CAMERA. It is now legit to call AiRender() without an explicit argument, which is much shorter to type. (#2613)

- **Added new AiArrayInterpolateXxx functions**: These new functions in the array API can be used to evaluate the value of an array element at a given time. (#2521)

```

AI_API AtPoint    AiArrayInterpolatePnt(const AtArray* array, float time, AtUInt32 idx);
AI_API AtVector   AiArrayInterpolateVec(const AtArray* array, float time, AtUInt32 idx);
AI_API float     AiArrayInterpolateFlt(const AtArray* array, float time, AtUInt32 idx);
AI_API void      AiArrayInterpolateMtx(const AtArray* array, float time, AtUInt32 idx, AtMatrix
result);

```

## Incompatible changes

- **Moved decay\_type to each of the lights:** The decay\_type parameter of the light\_decay filter has been moved to those light source nodes where it makes sense to have decay: point\_light, spot\_light, quad\_light, disk\_light and cylinder\_light. This makes it no longer necessary to create a light filter node and attach it to the filters parameter of the light if all you needed to change is the decay type. (#2457)
- **Changed default light decay to quadratic:** Previous default was constant, so existing scenes may need this value to be explicitly changed in order to render correctly. (#2509)
- **Removed linear and cubic light decay types:** These two legacy decay types don't make sense in a physically-based system. The only two supported decay types are now quadratic and constant. Note that certain critical features of the renderer such as MIS only work with quadraticdecay. (#2459)
- **Removed decay\_clamp and decay\_radius from light\_decay filter:** In an effort to simplify the system, we are removing these rarely used parameters. In particular, the decay\_clamp parameter is of little help now that the lights all have multiple importance sampling (MIS) support. (#2340)
- **Changed default specular BRDF in standard shader to cook\_torrance:** The previous default for standard.specular\_brdf was stretched\_phong, which is an older BRDF suffering from energy loss at grazing angles. Scenes with shaders that didn't explicitly set specular\_brdf may now render different. (#2548)
- **Fixed const-correctness in several API functions:** We are taking the opportunity to enforce const-correctness in a few places like AiMakeRay(), AiCameraGetLocalData()
(), AiNodeGet\*(), AiNodeGetLink(), AiNodeClone(), AiArrayConvert(), DriverSupportsPixelType(), AiDriverGetLocalData(), FilterOutputType(), AiFilterGetLocalData(), AiM4\*, AiAOVGet\*(), AiShaderEvalParam\*(), AiMetaDataIteratorFinished(). (#1882, #2238)
- **Hide AtNode implementation:** The contents of the AtNode struct are now hidden from the public API. Some new API functions have been added to provide access to hidden data (e.g. AiNodeGetNodeEntry(), see the API additions section). (#2057)
- **AiNodeLink() and AiNodeUnlink() now return a success value:** The new signatures are: (#2485)

```

AI_API bool AiNodeLink(AtNode* src, const char* param, AtNode* dest);
AI_API bool AiNodeUnlink(AtNode* node, const char* param);

```

- **Added comp argument to AiNodeGetLink():** This new parameter is used to return the output component selected in the source node. If no particular component was selected, -1 is returned meaning "the whole output"; if a single component was selected, then a value in 0..3 is returned. This parameter has a default value of NULL which means that the return value is not requested. Other than requiring a recompile, existing client code will be unaffected by this change. (#2052)
- **AI\_API AtNode\* AiNodeGetLink(const AtNode\* node, const char\* input, int\* comp = NULL);**
- **Added tid argument to camera\_create\_ray:** The persp\_camera node was sometimes crashing when using UV remapping because the camera\_create\_ray method wasn't thread-safe. This has been fixed by explicitly passing the thread ID as an argument. (#2499)
- **Changed Cubic BSSRDF signature:** (#2532) Since a new Gaussian BSSRDF was added, the old lookup API AiSSSPointCloudLookup() has been removed and renamed to:

```

AI_API AtColor AiSSSPointCloudLookupCubic(const AtShaderGlobals* sg, const AtColor& radius);

```

- **Changed AiRender() return type from unsigned int to int:** Because one of the return values of this function was -1 (AI\_ERROR) it doesn't make sense to return unsigned int. (#2482)
- **Modified math for noise.amplitude:** The output of the shader now converges to a value of 0.0 when the amplitude goes to 0.0 (it previously converged to 0.5). (#2489)
- **Renamed AiParameterXXX API to AiParameterXxx:** This was done for consistency with the AiNodeGetXxx() functions. (#2470)
- **Removed mis parameter from all lights:** For optimal results, multiple importance sampling should always be enabled, so the per-light mis parameter has been removed. We have added a ignore\_mis global setting to ignore MIS for all lights for debugging/benchmarking purposes. (#2510)
- **Removed AiNodeInstall() and AiNodeUninstall():** Use AiNodeEntryInstall() and AiNodeEntryUninstall() instead. (#2068)
- **Removed sss\_time:** Use reference\_time instead, see the Enhancements section above. (#2447)
- **Removed AtFloat and AtDouble from the public API:** These were silly types. Use float and double instead. Note that the renderer works with single-precision floats internally, and in previous releases we have already removed unnecessary uses of double-precision in the public API. (#2467, #2468)
- **Removed AtVoid from the public API:** Use void instead. (#2466)
- **Removed AtChar from the public API:** Use char instead. (#2504)
- **Removed AtBoolean from the public API:** AtBoolean is now deprecated, and all uses of it in the public API have been removed. You should use bool instead. Similarly, the TRUE and FALSE macros have been deprecated in favor of the more standard true and false. (#2507)
- **Removed AtInt and AtUInt from the public API:** Use int and unsigned int instead. (#2506)

- **Removed AtLong and AtULong from the public API:** These types had a different byte size in different platforms and therefore caused subtle bugs. You should use the AtUInt32 or AtUInt64 types instead, depending on what size you need. (#2469)
- **Removed AiTextureAccessResized():** Use the new AtTextureParams fields width\_s and width\_t instead. (#2458)
- **Removed legacy BRDFs:** Some of our oldest BRDFs that don't work well in a modern physically-based system have been removed: (#2232)
  - AiTorranceSparrowBRDF()
  - AiBlinnPhongBRDF()
  - AiPhongBRDF()
  - AiModifiedPhongBRDF()
  - AiLafortuneLobeBRDF()
  - AiModifiedPhongIntegrate()
  - AiLafortuneLobeIntegrate()
  - AiLafortuneLobeGetDir()
- **Removed make\_local\_copy parameter from AiArrayConvert:** This parameter is now effectively hardcoded to true. AtArray always owns its data memory (it allocates and deallocates the memory itself). The corresponding boolean in the AtArray structure has also been removed. (#2501)
- **Removed AiShaderGlobalsCalculateNormal() and AiShaderGlobalsGetBumpStates():** These functions have been deprecated for quite some time. Use AiShaderGlobalsEvaluateBump() instead. (#2282)
- **Removed diffuse AOV from lambert shader:** The lambert shader no longer exports the diffuse AOV and instead follows the example of the standard shader and exports the composable direct\_diffuse and indirect\_diffuse AOVs. (trac#2610)
- **Corrected parameter types in the AtArray API:** Some parameter types have been changed to better reflect the actual type of the data being passed. The number of elements is now limited to AtUInt32, and the number of motion keys to AtByte. The type parameter is now an AtByte. In any case, the total number of items (number of elements times number of keys) is limited to  $2^{32}$ . (#2514)
- **Changed return type of AtBRDFEvalBrdfFunc to AtColor:** The return type of AtBRDFEvalBrdfFunc functions used for MIS sampling via the AiEvaluateLightSample() API function has been changed from float to AtColor. As a result, the previous method of using chromatic BRDFs with MIS by setting the ARNOLD\_CHROMATIC\_BRDF environment variable is now obsolete and has been removed. (#2229)
- **Renamed dOdx and dOdy in AtShaderGlobals:** Both the naming and the comments for those data members in AtShaderGlobals were very confusing. They implied that these fields hold the derivatives of the ray origin, but actually it's the derivatives of the surface position being shaded. They are now renamed to dPdx and dPy. (#2552)
- **Added parameters to AiRadiance() and AiIrradiance():** These functions now receive a thread ID and a point ID, which allows them to be used in multi-threaded code. The new API looks like this: (#2156)

```
AI_API AtColor AiIrradiance(const AtPoint* p, const AtVector* n, AtByte tid, AtUInt32 pid);
AI_API AtColor AiRadiance(const AtPoint* p, const AtVector* dir, const AtVector* n, AtNode* obj,
AtUInt32 face, float u, float v, AtNode* shader, AtByte tid, AtUInt32 pid);
```

- **Removed AiRadianceSG() and AiIrradianceSG():** These functions are obsolete, as the fixes have been incorporated into AiRadiance and AiIrradiance. (#2156)
- **Replaced macros with inline functions in the API:** All the existing macros in the public API have been replaced by inlined functions, to allow for better error checking and help with performance by avoiding redundant evaluation of expressions. While most of these changes are transparent to the user, there have been some incompatible changes that affect client code: (#2518, #2617)
  - In general, for functions with more than one parameter, the use of a single type might force some client code to add an explicit type cast. For example, if there was a call like MAX(1, 2.0f), you will now have to manually resolve the type ambiguity, by using either MAX(1, 2) for integers, or MAX(1.0f, 2.0f) for floats. These changes might seem annoying at first, but they will give you better control over your code, and the compiler will be able to do better optimization.
  - SWAP() now receives 2 parameters instead of 3 (you no longer have to pass a temporary variable). Note that we provide SWAP() for legacy reasons only, and std::swap could be used instead.
  - AiFaceViewer() now takes 2 parameters: the shaderglobals struct pointer, and the normal vector to calculate. Previously, the macro was sneakily passing sg.

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#1610	exception crash when reading textures from non-existing drives	oiio	angel	critical	3.0.5	2 years
#2350	improve irradiance baking for SSS pointclouds from a shader	arnold	alan	critical	3.3	5 months
#2455	AOV blending through semi-opaque surfaces w/auto-transparency	arnold	alan	critical	3.3	3 months
#2473	'AiUDataGet*'() doesn't work with 'constant' user data in linux/darwin	arnold	oscar	critical	3.3	2 months
#2574	Flickering with "blue_noise_Pref" SSS sample distribution	arnold	oscar	critical	3.3	4 weeks
#2378	Inconsistency when referencing non-existent shaders	arnold	oscar	major	3.3	4 months
#2424	PNG associateAlpha + gamma lost all precision	oiio	thiago	major	3.3	3 months
#2440	precision problem in point_light's light_trace	arnold	oscar	major	3.3	3 months
#2446	Math typo when computing tangents in subdiv code	arnold	oscar	major	3.3	3 months
#2449	AiTTextureGet*() doesn't take ignore_textures into account	arnold	oscar	major	3.3	3 months
#2463	fix documentation errors in world/camera/screen matrix API	arnold	marcos	major	3.3	2 months
#2465	Node names containing spaces should be wrapped in quotes when written to .ass	arnold	angel	major	3.3	2 months
#2474	light samples sometimes traveling through solid objects	arnold	alan	major	3.3	2 months
#2477	Procedural error when either 'min' or 'max' is not specified	arnold	angel	major	3.3	2 months
#2486	Wrong reported lines in '.ass' parsing warnings/errors	arnold	oscar	major	3.3	2 months
#2487	fix double warning when parsing unknown parameters in .ass files	arnold	oscar	major	3.3	2 months
#2489	Bug in the amplitude parameter of the built-in noise shader	arnold	angel	major	3.3	2 months
#2492	poor multi-threading performance when autotile is enabled	oiio	ramon	major	3.3	2 months
#2493	Better '.ass' parsing for mismatched number of expected values	arnold	oscar	major	3.3	2 months
#2494	crash with forward references in light_group	arnold	oscar	major	3.3	2 months
#2497	Crash when connecting 'volume_scattering' to light's 'volume_density'	arnold	oscar	major	3.3	8 weeks
#2499	persp_camera UV distortion not thread safe	arnold	oscar	major	3.3	8 weeks
#2513	shadowing options are not overridden in instances	arnold	oscar	major	3.3	7 weeks
#2522	hair diffuse cache brighter than expected (not energy conserving)	arnold	marcos	major	3.3	7 weeks
#2525	hair diffuse cache not affected by the light's affect_diffuse parameter	arnold	marcos	major	3.3	7 weeks
#2526	hair shader is hardcoded to 1 bounce when using the diffuse cache	arnold	marcos	major	3.3	7 weeks
#2528	Support varying matrix user data	arnold	oscar	major	3.3	7 weeks
#2535	Cubic BSSRDF normalization is incorrect	arnold	marcos	major	3.3	6 weeks
#2544	motion-blurred subdivision surface not correctly computing limit normals at boundaries	arnold	alan	major	3.3	6 weeks
#2554	kick -af is missing support for blackman_harris filter	kick	marcos	major	3.3	5 weeks
#2566	crash/hang with consecutive SIGINT/SIGTERM	arnold	oscar	major	3.3	5 weeks
#2575	cylinder and disk lights conspiring to kill each other off	arnold	oscar	major	3.3	4 weeks
#2579	crash with subdivision and motion-blur	arnold	marcos	major	3.3	4 weeks
#2584	erroneous composition of alpha value of atmosphere shaders	arnold	alan	major	3.3	3 weeks
#2597	Cloning of nodes inside procedurals is not working	arnold	angel	major	3.3	13 days
#2598	pykick is broken after `AiASSLoad()` changes	kick	oscar	major	3.3	13 days
#2601	Replace use of non-standard __declspec with __declspec	arnold	angel	major	3.3	12 days
#2600	points are not recording indices memory in stats	arnold	thiago	minor	3.3	13 days

## 3.x

- 3.3.13.2
- 3.3.13.1
- 3.3.13.0
- 3.3.12.1
- 3.3.12.0
- 3.3.11.1
- 3.3.11.0
- 3.3.10.2
- 3.3.10.1
- 3.3.10.0
- 3.3.9.0
- 3.3.8.0
- 3.3.7.0
- 3.3.6.0
- 3.3.5.0
- 3.3.4.2
- 3.3.3.2
- 3.3.2.2
- 3.3.0.0
- 3.3.1.0
- 3.2.5.0
- 3.2.4.0
- 3.2.3.1
- 3.2.2.0
- 3.2.1.0
- 3.2.0.0
- 3.1.1.0

### **3.3.13.2**

## Milestone 3.3.13

## Enhancements

- **Binary encoding of float arrays in .ass files:** A new type of encoding is used to compress large float arrays into a more compact ASCII representation, leading to smaller files and faster load times. In addition, the new encoding has exact 32-bit precision, whereas previously we truncated and stored the floats into at most 8 ASCII digits (e.g. 1234.5678). The encoded arrays are indicated by prefixing the array type with "b64" as in the example below. By default, Arnold will write encoded .ass files, but this can be turned off with the new option binary\_ass. (trac#2434)

- **Statistics for largest meshes:** The triangle tessellation portion of the statistics output now includes a list of the top 5 heaviest meshes by triangle count. This can be used to quickly identify over-tesselated meshes in your scenes. (trac#2438)
  - **sss\_threaded\_sample\_distribution = false disables binning in SSS point clouds:** As a work-around that ensures that any possible banding artifacts will be removed from the blue-noise SSS distributions, the sss\_threaded\_sample\_distribution will cause the renderer to avoid applying the binning multi-threading technique that was sometimes causing banding artifacts in SSS samples. This of course comes at the cost of disabling multi-threaded sample generation. (#2426)
  - **utility shader can visualize sg->Ns:** We have added ns to the utility.color\_mode enum to visualize sg->Ns (the smooth, un-bumped normal). This complements the existing ng and n modes. (trac#2425)
  - **linear interpolation on curves:** The curve primitive now supports a linear interpolation type that does not require a minimum of 4 control points. Note that the curve is still rendered as a cubic curve internally, so the appearance may be slightly different compared to PRMan. (trac#2431)
  - **Added raw motion vector support:** A raw parameter has been added to the motion\_vector shader. This mode stores the raw, un-encoded motion vector directly in the RG components of the shader's output. The default value is FALSE. (trac#2432)
  - **Added motionvector built-in AOV:** This AOV provides raw, un-encoded 2D motion vectors between shutter start and end. Note that the calculation of motion vectors is slow. To render an image with motion vectors but no 3D motion blur, you may use the ignore\_motion\_blur global option. For best results, this AOV should be filtered similarly to Z values. (trac#2433)
  - **Added Pref built-in AOV:** Arnold can now output Pref coordinates (sometimes called pose reference coordinates) to an AOV, similar to the P AOV. This would only work in scenes that have been exported with Pref user-data. (trac#2435)

## API additions

- **AiSSSTraceSingleScatter()**: This new function gives a single scattered approximation to SSS along the ray direction. The number of samples taken to compute the result is controlled linearly by the new GI\_single\_scatter\_samples render option. This function uses pretty much the same parameterization as Wann Jensen's 2001 BSSRDF paper, where the mean free path parameter can be computed for the values from the table in that paper using the following formula:  $mfp = 1/\sigma_{t'} = 1/(\sigma_s' + \sigma_a)$ . Note that the effect of this function can be disabled with the ignore\_sss option. (#2169)
  - **AI\_LOG\_SSS**: Added new log type AI\_LOG\_SSS, applied to SSS messages so that they are moved up to level 5 verbosity in kick. (trac#2421)

## Incompatible changes

- **AiLoadPlugin()**: The arguably redundant AiLoadPlugin() function is now deprecated. Its functionality has been added to AiLoadPlugins(), which can now load any combination of directories and explicit plugin files, separated by ':' in Linux/OSX or ';' in Windows. The deprecated function will be removed in the next major release. (trac#2419)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2391	strong bump mapping produces specular faceting artifacts	arnold	alan	critical	3.3	3 months
#2416	bump mapping epsilon is too big for some scenes	arnold	alan	critical	3.3	2 months
#2405	reduce artifacts at bin boundaries in blue noise SSS point cloud construction	arnold	oscar	major	3.3	3 months
#2408	reduce numerical instability in pixel-length estimate for adaptive subdivision	arnold	marcos	major	3.3	3 months
#2412	'#' should be allowed in quoted string attributes	arnold	ramon	major	3.3	3 months
#2418	Plugins with .sog extension are not automatically loaded	arnold	angel	major	3.3	2 months
#2422	sg->Ns should not include autobump	arnold	ramon	major	3.3	2 months
#2426	Fix dark lines in SSS when 'sss_threaded_sample_distribution' is false	arnold	oscar	major	3.3	2 months
#2428	Crash when using a shader which is not a shader	arnold	oscar	major	3.3	2 months
#2442	polygon_midpoint SSS skipping faces of axis aligned grids	arnold	alan	major	3.3	2 months

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2465	Node names containing spaces should be wrapped in quotes when written to .ass	3.3.13.1	arnold	angel	major	3.4
#2474	light samples sometimes traveling through solid objects	mill, 3.3.13.1	arnold	alan	major	3.4

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2574	Flickering with "blue_noise_Pref" SSS sample distribution	dd, 3.3.13.2	arnold	oscar	critical	3.4

### **3.3.13.1**

Milestone 3.3.13

## Enhancements

- **Binary encoding of float arrays in .ass files:** A new type of encoding is used to compress large float arrays into a more compact ASCII representation, leading to smaller files and faster load times. In addition, the new encoding has exact 32-bit precision, whereas previously we truncated and stored the floats into at most 8 ASCII digits (e.g. 1234.5678). The encoded arrays are indicated by prefixing the array type with "b64" as in the example below. By default, Arnold will write encoded .ass files, but this can be turned off with the new option binary\_ass. (trac#2434)

- **Statistics for largest meshes:** The triangle tessellation portion of the statistics output now includes a list of the top 5 heaviest meshes by triangle count. This can be used to quickly identify over-tesselated meshes in your scenes. (trac#2438)
  - **sss\_threaded\_sample\_distribution = false disables binning in SSS point clouds:** As a work-around that ensures that any possible banding artifacts will be removed from the blue-noise SSS distributions, the sss\_threaded\_sample\_distribution will cause the renderer to avoid applying the binning multi-threading technique that was sometimes causing banding artifacts in SSS samples. This of course comes at the cost of disabling multi-threaded sample generation. (#2426)
  - **utility shader can visualize sg->Ns:** We have added ns to the utility.color\_mode enum to visualize sg->Ns (the smooth, un-bumped normal). This complements the existing ng and n modes. (trac#2425)
  - **linear interpolation on curves:** The curve primitive now supports a linear interpolation type that does not require a minimum of 4 control points. Note that the curve is still rendered as a cubic curve internally, so the appearance may be slightly different compared to PRMan. (trac#2431)
  - **Added raw motion vector support:** A raw parameter has been added to the motion\_vector shader. This mode stores the raw, un-encoded motion vector directly in the RG components of the shader's output. The default value is FALSE. (trac#2432)
  - **Added motionvector built-in AOV:** This AOV provides raw, un-encoded 2D motion vectors between shutter start and end. Note that the calculation of motion vectors is slow. To render an image with motion vectors but no 3D motion blur, you may use the ignore\_motion\_blur global option. For best results, this AOV should be filtered similarly to Z values. (trac#2433)
  - **Added Pref built-in AOV:** Arnold can now output Pref coordinates (sometimes called pose reference coordinates) to an AOV, similar to the P AOV. This would only work in scenes that have been exported with Pref user-data. (trac#2435)

## API additions

- **AiSSSTraceSingleScatter()**: This new function gives a single scattered approximation to SSS along the ray direction. The number of samples taken to compute the result is controlled linearly by the new GI\_single\_scatter\_samples render option. This function uses pretty much the same parameterization as Wann Jensen's 2001 BSSRDF paper, where the mean free path parameter can be computed for the values from the table in that paper using the following formula:  $mfp = 1/\sigma_t' = 1/(\sigma_s + \sigma_a)$ . Note that the effect of this function can be disabled with the ignore\_sss option. (#2169)
  - **AI\_LOG\_SSS**: Added new log type AI\_LOG\_SSS, applied to SSS messages so that they are moved up to level 5 verbosity in kick. (trac#2421)

## Incompatible changes

- **AiLoadPlugin()**: The arguably redundant AiLoadPlugin() function is now deprecated. Its functionality has been added to AiLoadPlugins(), which can now load any combination of directories and explicit plugin files, separated by ':' in Linux/OSX or ';' in Windows. The deprecated function will be removed in the next major release. (trac#2419)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2391	strong bump mapping produces specular faceting artifacts	arnold	alan	critical	3.3	8 weeks
#2416	bump mapping epsilon is too big for some scenes	arnold	alan	critical	3.3	5 weeks
#2405	reduce artifacts at bin boundaries in blue noise SSS point cloud construction	arnold	oscar	major	3.3	6 weeks
#2408	reduce numerical instability in pixel-length estimate for adaptive subdivision	arnold	marcos	major	3.3	6 weeks
#2412	'#' should be allowed in quoted string attributes	arnold	ramon	major	3.3	6 weeks
#2418	Plugins with .sog extension are not automatically loaded	arnold	angel	major	3.3	5 weeks
#2422	sg->Ns should not include autobump	arnold	ramon	major	3.3	5 weeks
#2426	Fix dark lines in SSS when 'sss_threaded_sample_distribution' is false	arnold	oscar	major	3.3	5 weeks
#2428	Crash when using a shader which is not a shader	arnold	oscar	major	3.3	5 weeks
#2442	polygon_midpoint SSS skipping faces of axis aligned grids	arnold	alan	major	3.3	4 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2465	Node names containing spaces should be wrapped in quotes when written to .ass	3.3.13.1	arnold	angel	major	3.4
#2474	light samples sometimes traveling through solid objects	mill, 3.3.13.1	arnold	alan	major	3.4

### 3.3.13.0

Milestone 3.3.13

## Enhancements

- **Binary encoding of float arrays in .ass files:** A new type of encoding is used to compress large float arrays into a more compact ASCII representation, leading to smaller files and faster load times. In addition, the new encoding has exact 32-bit precision, whereas previously we truncated and stored the floats into at most 8 ASCII digits (e.g. 1234.5678). The encoded arrays are indicated by prefixing the array type with "b64" as in the example below. By default, Arnold will write encoded .ass files, but this can be turned off with the new option binary\_ass. (trac#2434)

- **Statistics for largest meshes:** The triangle tessellation portion of the statistics output now includes a list of the top 5 heaviest meshes by triangle count. This can be used to quickly identify over-tesselated meshes in your scenes. (trac#2438)
  - **sss\_threaded\_sample\_distribution = false disables binning in SSS point clouds:** As a work-around that ensures that any possible banding artifacts will be removed from the blue-noise SSS distributions, the sss\_threaded\_sample\_distribution will cause the renderer to avoid applying the binning multi-threading technique that was sometimes causing banding artifacts in SSS samples. This of course comes at the cost of disabling multi-threaded sample generation. (#2426)
  - **utility shader can visualize sg->Ns:** We have added ns to the utility.color\_mode enum to visualize sg->Ns (the smooth, un-bumped normal). This complements the existing ng and n modes. (trac#2425)
  - **linear interpolation on curves:** The curve primitive now supports a linear interpolation type that does not require a minimum of 4 control points. Note that the curve is still rendered as a cubic curve internally, so the appearance may be slightly different compared to PRMan. (trac#2431)
  - **Added raw motion vector support:** A raw parameter has been added to the motion\_vector shader. This mode stores the raw, un-encoded motion vector directly in the RG components of the shader's output. The default value is FALSE. (trac#2432)
  - **Added motionvector built-in AOV:** This AOV provides raw, un-encoded 2D motion vectors between shutter start and end. Note that the calculation of motion vectors is slow. To render an image with motion vectors but no 3D motion blur, you may use the ignore\_motion\_blur global option. For best results, this AOV should be filtered similarly to Z values. (trac#2433)
  - **Added Pref built-in AOV:** Arnold can now output Pref coordinates (sometimes called pose reference coordinates) to an AOV, similar to the P AOV. This would only work in scenes that have been exported with Pref user-data. (trac#2435)

## API additions

- **AiSSSTraceSingleScatter()**: This new function gives a single scattered approximation to SSS along the ray direction. The number of samples taken to compute the result is controlled linearly by the new GI\_single\_scatter\_samples render option. This function uses pretty much the same parameterization as Wann Jensen's 2001 BSSRDF paper, where the mean free path parameter can be computed for the values from the table in that paper using the following formula:  $mfp = 1/\sigma_{t'} = 1/(\sigma_s' + \sigma_a)$ . Note that the effect of this function can be disabled with the ignore\_sss option. (#2169)
  - **AI\_LOG\_SSS**: Added new log type AI\_LOG\_SSS, applied to SSS messages so that they are moved up to level 5 verbosity in kick. (trac#2421)

## Incompatible changes

- **AiLoadPlugin()**: The arguably redundant AiLoadPlugin() function is now deprecated. Its functionality has been added to AiLoadPlugins(), which can now load any combination of directories and explicit plugin files, separated by ':' in Linux/OSX or ';' in Windows. The deprecated function will be removed in the next major release. (trac#2419)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2391	strong bump mapping produces specular faceting artifacts	arnold	alan	critical	3.3	4 weeks
#2416	bump mapping epsilon is too big for some scenes	arnold	alan	critical	3.3	11 days
#2405	reduce artifacts at bin boundaries in blue noise SSS point cloud construction	arnold	oscar	major	3.3	2 weeks
#2408	reduce numerical instability in pixel-length estimate for adaptive subdivision	arnold	marcos	major	3.3	2 weeks
#2412	'#' should be allowed in quoted string attributes	arnold	ramon	major	3.3	2 weeks
#2418	Plugins with .sog extension are not automatically loaded	arnold	angel	major	3.3	10 days
#2422	sg->Ns should not include autobump	arnold	ramon	major	3.3	10 days
#2426	Fix dark lines in SSS when 'sss_threaded_sample_distribution' is false	arnold	oscar	major	3.3	8 days
#2428	Crash when using a shader which is not a shader	arnold	oscar	major	3.3	7 days
#2442	polygon_midpoint SSS skipping faces of axis aligned grids	arnold	alan	major	3.3	29 hours

### 3.3.12.1

#### Milestone 3.3.12

##### Enhancements

- **Added disp\_padding to set displacement bounds:** The displacement bounds are now set with the disp\_padding attribute of the polymesh node. Previously, this was set via the disp\_height parameter, which was also used to set the amplitude of displacement, causing confusion and even rendering artifacts in some cases. (trac#2399).
- **Adaptive subdivision fixes:** A bug has been fixed in the flatness-based adaptive subdivision code, and as a result the renderer now produces more stable tessellations in animation, and more accurately reaches the target subdiv\_pixel\_error set by the user. This has the apparent effect of reducing the number of polygons, because the old, buggy code was over-tessellating. (trac#2330)

##### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2330	adaptive subdivision artifact in flatness mode	arnold	alan	major	3.3	3 months
#2393	alpha channel shouldn't have random dithering in 8/16-bit output	arnold	marcos	major	3.3	4 weeks
#2396	NaNs coming from `AiLightsGetShadowMatte()`	arnold	alan	major	3.3	4 weeks
#2398	procedural expansion time stats are broken	arnold	marcos	major	3.3	4 weeks
#2402	SSS log messages should not appear at minimum verbosity	arnold	marcos	trivial	3.3	3 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2405	reduce artifacts at bin boundaries in blue noise SSS point cloud construction	3.3.12.1	arnold	oscar	major	3.3.13
#2412	'#' should be allowed in quoted string attributes	ass parsing, 3.3.12.1	arnold	ramon	major	3.3.13
#2442	polygon_midpoint SSS skipping faces of axis aligned grids	3.3.12.1	arnold	alan	major	3.3.13

## 3.3.12.0

### Milestone 3.3.12

#### Enhancements

- **Added disp\_padding to set displacement bounds:** The displacement bounds are now set with the disp\_padding attribute of the polymesh node. Previously, this was set via the disp\_height parameter, which was also used to set the amplitude of displacement, causing confusion and even rendering artifacts in some cases. (trac#2399).
- **Adaptive subdivision fixes:** A bug has been fixed in the flatness-based adaptive subdivision code, and as a result the renderer now produces more stable tessellations in animation, and more accurately reaches the target subdiv\_pixel\_error set by the user. This has the apparent effect of reducing the number of polygons, because the old, buggy code was over-tessellating. (trac#2330)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2330	adaptive subdivision artifact in flatness mode	arnold	alan	major	3.3	2 months
#2393	alpha channel shouldn't have random dithering in 8/16-bit output	arnold	marcos	major	3.3	2 weeks
#2396	NaNs coming from `AiLightsGetShadowMatte()`	arnold	alan	major	3.3	12 days
#2398	procedural expansion time stats are broken	arnold	marcos	major	3.3	10 days
#2402	SSS log messages should not appear at minimum verbosity	arnold	marcos	trivial	3.3	5 days

### 3.3.11.1

#### Milestone 3.3.11

##### Enhancements

- **Simplified SSS sampling controls:** To simplify the user experience of dialing light and GI samples for SSS, we have unified all the controls to be controlled by a global multiplier: `sss_sample_factor`. This control behaves similarly to `AA_samples`, but in the SSS context, multiplying against the light/diffuse sampling settings the user has already dialed for his scene. The default value is 4. Typically it should be set to the same as `AA_samples` to ensure every point in the point cloud receives noise-free lighting. The per-light `sss_samples` and the global `GI_sss_hemi_samples` controls have been removed. (trac#2364)
- **Added option `sss_threaded_sample_distribution`:** This is a temporary "escape valve" to disable multi-threading in the construction of SSS pointcloud distributions. Although we are confident that most of the recent SSS bugs were solved in this release, we are providing this option for users who may encounter additional threading hangs/crashes. The default value is TRUE, consistent with previous behaviour. This option will be removed in a future release. (trac#2387)
- **New `disk_light`:** We have added a `disk_light` node that implements an oriented disk light. The new light supports all of the standard features such as MIS and volumetrics. (trac#2197)
- **MIS support in `AiDirectDiffuse()`:** The `AiDirectDiffuse()` function available to shader writers and used by the built-in lambert shader will now apply multiple importance sampling techniques when available. This can noticeably improve image quality, in particular with skylight lights. (trac#2313)
- **multiply and offset in image shader:** We have added two handy controls in the image node. Thanks to these new controls, there is no need to write a separate shader when you simply need to filter the texture lookup by a color, or add a color to the texture lookup. (trac#2352)
- **Support for Pref coordinates in noise shader:** In addition to the existing world and object coordinate spaces, it is now possible to evaluate the noise at Pref coordinates. This of course assumes that the mesh has been exported with Pref data, otherwise it reverts to the default objectspace. (trac#2348)
- **Set `@executable_path` in Mac OSX binaries:** On Mac OSX systems, the paths to libraries used by Arnold binaries are now hardcoded to `@executable_path`. This makes libai to always link first with libraries residing in the directory where libai itself is located, avoiding loading libraries with mismatched versions that may be reached through the `DYLD_LIBRARY_PATH` environment variable. It is now ensured that libai links with the libraries deployed in the official Arnold package as both libai and its dependencies are located in the package directory tree. Therefore we encourage users not to relocate the Arnold binaries outside the `<arnold_root_dir>/bin` directory. (trac#2223)
- **Floating point support in `driver_display`:** The generic display driver node `driver_display` receives RGB/RGBA buckets and sends the pixel data to a custom C-style callback. You can now choose the format for those pixel buffers between the original floating point data (with gamma correction applied) or the old "packed integer" format which is 8 bits per channel (with clamping to 0-1, gamma correction and dithering applied). This is done using the new `rgba_packing` boolean parameter, which defaults to TRUE to maintain previous behaviour. (trac#2175)
- **Added option `-sr <f>` to kick:** The new kick option `-sr <f>` allows to scale up/down `<f>` times the resolution of the output image. (trac#2347)

##### API additions

- **Python bindings for the Licensing API:** These bindings allow configuring the licensing subsystem (server, number of attempts and delay). (trac#2237)

##### Incompatible changes

- **Removed SSS sampling controls:** The global option `GI_sss_hemi_samples` and the per-light option `sss_samples` have been removed in favor of the new global option `sss_sample_factor` described above. Note that this can change the quality of the SSS effect in existing scenes that were using these options to override the SSS samples, and therefore render times can be affected too. If, after upgrading to this version, your renders suddenly become very slow, you may want to reduce `sss_sample_factor`; conversely, if your renders suddenly become much faster and noisier, try increasing `sss_sample_factor`. (trac#2364)
- **Removed `sss_use_gi` from objects:** As part of an ongoing effort to simplify the SSS system, we have removed the `sss_use_gi` geometric object attribute. The SSS engine will now always evaluate indirect lighting, with the same number of bounces as specified in the `GI_diffuse_depth` option. This can result in differences in rendered images for older scenes that had this parameter disabled, but these cases should be rare. As an added bonus, this saves memory in scenes with many objects. (trac#2363)
- **Removed option `-qres` from kick:** You can achieve the same effect with the new option `-sr 0.5`. (trac#2347)
- **Renamed the disc primitive to disk:** For better consistency with other node/parameter strings in the API (i.e. the `disk_light` and the disk mode of the points primitive), the rarely-used disc geometric primitive has been renamed to disk. A deprecated synonym has been added so that the old name still works (at the expense of a warning message). (trac#2357)
- **Removed the ability to set thread priorities on Linux/OSX:** For a long time, this feature has been broken on Linux and OSX, so we are now officially removing support for those platforms. You can use the Unix nice command to run the Arnold process at lower priority. `AiThreadCreate()`'s third argument is still accepted, but only has an effect on Windows, where it's perfectly fine to alter the

priority of render threads. (trac#1077)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2100	AiArray(Set Get){Vec Pnt} should work on both points and vectors	arnold	oscar	major	3.3	7 months
#2173	hair direct diffuse and direct specular brighter than expected	arnold	alan	major	3.3	5 months
#2308	AiRGBACreate() should return an AtRGBABlend	arnold	oscar	major	3.3	2 months
#2355	SSS pointcloud build crashes on disjoint meshes	arnold	xo	major	3.3	5 weeks
#2359	Livelock in SSS pointcloud building	arnold	xo	major	3.3	5 weeks
#2361	SSS pointcloud building causes render to hang	arnold	oscar	major	3.3	5 weeks
#2370	'kick -info [n u] <node>' doesn't work	kick	oscar	major	3.3	4 weeks
#2372	Texture blurriness when scaling cameras	arnold	oscar	major	3.3	4 weeks
#2375	Aborting when constructing SSS pointclouds causes render to crash/hang	arnold	oscar	major	3.3	4 weeks
#2380	`AiLightsGetShadowMatte()` missing some shadows	arnold	alan	major	3.3	3 weeks
#2389	'bounces' control on the lights should affect glossy bounces	arnold	marcos	major	3.3	8 days
#2358	Spinlock implementation fails on newer version of GCC due to optimization	arnold	xo	minor	3.3	5 weeks
#2373	'kick' doesn't work properly with '-interactive [m q]' and camera matrices	kick	oscar	minor	3.3	4 weeks
#2365	'disk' count is not reported in the initial log messages	arnold	oscar	trivial	3.3	5 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2396	NaNs coming from `AiLightsGetShadowMatte()`	3.3.11.1	arnold	alan	major	3.3.12

### 3.3.11.0

#### Milestone 3.3.11

##### Enhancements

- **Simplified SSS sampling controls:** To simplify the user experience of dialing light and GI samples for SSS, we have unified all the controls to be controlled by a global multiplier: `sss_sample_factor`. This control behaves similarly to `AA_samples`, but in the SSS context, multiplying against the light/diffuse sampling settings the user has already dialed for his scene. The default value is 4. Typically it should be set to the same as `AA_samples` to ensure every point in the point cloud receives noise-free lighting. The per-light `sss_samples` and the global `GI_sss_hemi_samples` controls have been removed. (trac#2364)
- **Added option `sss_threaded_sample_distribution`:** This is a temporary "escape valve" to disable multi-threading in the construction of SSS pointcloud distributions. Although we are confident that most of the recent SSS bugs were solved in this release, we are providing this option for users who may encounter additional threading hangs/crashes. The default value is TRUE, consistent with previous behaviour. This option will be removed in a future release. (trac#2387)
- **New `disk_light`:** We have added a `disk_light` node that implements an oriented disk light. The new light supports all of the standard features such as MIS and volumetrics. (trac#2197)
- **MIS support in `AiDirectDiffuse()`:** The `AiDirectDiffuse()` function available to shader writers and used by the built-in lambert shader will now apply multiple importance sampling techniques when available. This can noticeably improve image quality, in particular with skylight lights. (trac#2313)
- **multiply and offset in image shader:** We have added two handy controls in the image node. Thanks to these new controls, there is no need to write a separate shader when you simply need to filter the texture lookup by a color, or add a color to the texture lookup. (trac#2352)
- **Support for Pref coordinates in noise shader:** In addition to the existing world and object coordinate spaces, it is now possible to evaluate the noise at Pref coordinates. This of course assumes that the mesh has been exported with Pref data, otherwise it reverts to the default objectspace. (trac#2348)
- **Set `@executable_path` in Mac OSX binaries:** On Mac OSX systems, the paths to libraries used by Arnold binaries are now hardcoded to `@executable_path`. This makes libai to always link first with libraries residing in the directory where libai itself is located, avoiding loading libraries with mismatched versions that may be reached through the `DYLD_LIBRARY_PATH` environment variable. It is now ensured that libai links with the libraries deployed in the official Arnold package as both libai and its dependencies are located in the package directory tree. Therefore we encourage users not to relocate the Arnold binaries outside the `<arnold_root_dir>/bin` directory. (trac#2223)
- **Floating point support in `driver_display`:** The generic display driver node `driver_display` receives RGB/RGBA buckets and sends the pixel data to a custom C-style callback. You can now choose the format for those pixel buffers between the original floating point data (with gamma correction applied) or the old "packed integer" format which is 8 bits per channel (with clamping to 0-1, gamma correction and dithering applied). This is done using the new `rgba_packing` boolean parameter, which defaults to TRUE to maintain previous behaviour. (trac#2175)
- **Added option `-sr <f>` to kick:** The new kick option `-sr <f>` allows to scale up/down `<f>` times the resolution of the output image. (trac#2347)

##### API additions

- **Python bindings for the Licensing API:** These bindings allow configuring the licensing subsystem (server, number of attempts and delay). (trac#2237)

##### Incompatible changes

- **Removed SSS sampling controls:** The global option `GI_sss_hemi_samples` and the per-light option `sss_samples` have been removed in favor of the new global option `sss_sample_factor` described above. Note that this can change the quality of the SSS effect in existing scenes that were using these options to override the SSS samples, and therefore render times can be affected too. If, after upgrading to this version, your renders suddenly become very slow, you may want to reduce `sss_sample_factor`; conversely, if your renders suddenly become much faster and noisier, try increasing `sss_sample_factor`. (trac#2364)
- **Removed `sss_use_gi` from objects:** As part of an ongoing effort to simplify the SSS system, we have removed the `sss_use_gi` geometric object attribute. The SSS engine will now always evaluate indirect lighting, with the same number of bounces as specified in the `GI_diffuse_depth` option. This can result in differences in rendered images for older scenes that had this parameter disabled, but these cases should be rare. As an added bonus, this saves memory in scenes with many objects. (trac#2363)
- **Removed option `-qres` from kick:** You can achieve the same effect with the new option `-sr 0.5`. (trac#2347)
- **Renamed the disc primitive to disk:** For better consistency with other node/parameter strings in the API (i.e. the `disk_light` and the disk mode of the points primitive), the rarely-used disc geometric primitive has been renamed to disk. A deprecated synonym has been added so that the old name still works (at the expense of a warning message). (trac#2357)
- **Removed the ability to set thread priorities on Linux/OSX:** For a long time, this feature has been broken on Linux and OSX, so we are now officially removing support for those platforms. You can use the Unix nice command to run the Arnold process at lower priority. `AiThreadCreate()`'s third argument is still accepted, but only has an effect on Windows, where it's perfectly fine to alter the

priority of render threads. (trac#1077)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2100	AiArray(Set Get){Vec Pnt} should work on both points and vectors	arnold	oscar	major	3.3	7 months
#2173	hair direct diffuse and direct specular brighter than expected	arnold	alan	major	3.3	5 months
#2308	AiRGBACreate() should return an AtRGBABlend	arnold	oscar	major	3.3	2 months
#2355	SSS pointcloud build crashes on disjoint meshes	arnold	xo	major	3.3	4 weeks
#2359	Livelock in SSS pointcloud building	arnold	xo	major	3.3	4 weeks
#2361	SSS pointcloud building causes render to hang	arnold	oscar	major	3.3	4 weeks
#2370	'kick -info [n u] <node>' doesn't work	kick	oscar	major	3.3	3 weeks
#2372	Texture blurriness when scaling cameras	arnold	oscar	major	3.3	3 weeks
#2375	Aborting when constructing SSS pointclouds causes render to crash/hang	arnold	oscar	major	3.3	3 weeks
#2380	`AiLightsGetShadowMatte()` missing some shadows	arnold	alan	major	3.3	11 days
#2389	'bounces' control on the lights should affect glossy bounces	arnold	marcos	major	3.3	16 hours
#2358	Spinlock implementation fails on newer version of GCC due to optimization	arnold	xo	minor	3.3	4 weeks
#2373	'kick' doesn't work properly with '-interactive [m q]' and camera matrices	kick	oscar	minor	3.3	3 weeks
#2365	'disk' count is not reported in the initial log messages	arnold	oscar	trivial	3.3	4 weeks

### 3.3.10.2

#### Milestone 3.3.10

##### Enhancements

- **Improved sampling of the skydome\_light with diffuse shading:** The MIS technique will now also be applied to the diffuse component of the standard shader when lit by skydome lights. This can result in a great reduction in noise in some cases, particularly for scenes lit by skydome lights that have low contrast maps applied. Note that this change also applies to all shaders that use the Oren-Nayar MIS API. (trac#2338)
- **Optimized skydome\_light and textured quad\_light importance sampling:** Shader color values for precomputing importance sampling are now stored alongside the importance table in a table of equal size. Using this table for lighting gives a dramatic speedup over re-evaluating the shader, particularly if the attached shader is expensive to evaluate. 2x and even 4x speedups are not uncommon. In most cases, the quality difference of this re-sampling is unnoticeable, and diminishes as skydome\_light.resolution and quad\_light.resolution increases. This feature can be disabled by setting enable\_fast\_importance\_tables to off. (trac#2342)

##### API additions

- **Metadata flag for non-linkable parameters:** Node parameters can be explicitly flagged as non-linkable by setting the following boolean metadata to false:

```
node_parameters
{
    ...
    AiMetaDataSetBool(mds, "parameter_name", "linkable", FALSE);
}
```

Attempts to use AiNodeLink() on a parameter that has been flagged as non-linkable will produce a warning but otherwise have no effect. (trac#2341)

- **Application defined string:** Client applications, such as the Maya and Softimage plugins, are now allowed to pass a descriptive string to Arnold using the following API function:

```
AI_API AtVoid AiSetAppString(const char *appstr);
```

The app string will be printed in the output log and in the header of exported .ass files, which can greatly help developers when debugging customer scenes. We have also provided a corresponding Python binding. (trac#2337)

##### Incompatible changes

- **Renamed GI\_direct\_lighting to ignore\_direct\_lighting:** The ignore\_direct\_lighting option makes more sense alongside the other ignore\_\* options because it is mainly used for troubleshooting. Note that this flips the meaning of the option, so it is an incompatible change, not a simple rename. Along with the renaming, the -dd option in kick and pykick has also been changed to -idirect. (trac#2335)

##### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#1268	default auto-naming of nodes is not thread-safe	arnold	xo	major	2.33	2 years
#2344	Race condition in importance table initialization	arnold	xo	major	3.3	6 weeks
#2345	AiNode (node creation) function is not thread-safe	arnold	xo	major	3.3	6 weeks
#2346	Data race in string pool add	arnold	xo	major	3.3	6 weeks

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2355	SSS pointcloud build crashes on disjoint meshes	3.3.10.1	arnold	xo	major	3.3.11
#2359	Livelock in SSS pointcloud building	3.3.10.1	arnold	xo	major	3.3.11

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2361	SSS pointcloud building causes render to hang	3.3.10.2	arnold	oscar	major	3.3.11
#2375	Aborting when constructing SSS pointclouds causes render to crash/hang	3.3.10.2	arnold	oscar	major	3.3.11

### 3.3.10.1

#### Milestone 3.3.10

##### Enhancements

- **Improved sampling of the skydome\_light with diffuse shading:** The MIS technique will now also be applied to the diffuse component of the standard shader when lit by skydome lights. This can result in a great reduction in noise in some cases, particularly for scenes lit by skydome lights that have low contrast maps applied. Note that this change also applies to all shaders that use the Oren-Nayar MIS API. (trac#2338)
- **Optimized skydome\_light and textured quad\_light importance sampling:** Shader color values for precomputing importance sampling are now stored alongside the importance table in a table of equal size. Using this table for lighting gives a dramatic speedup over re-evaluating the shader, particularly if the attached shader is expensive to evaluate. 2x and even 4x speedups are not uncommon. In most cases, the quality difference of this re-sampling is unnoticeable, and diminishes as skydome\_light.resolution and quad\_light.resolution increases. This feature can be disabled by setting enable\_fast\_importance\_tables to off. (trac#2342)

##### API additions

- **Metadata flag for non-linkable parameters:** Node parameters can be explicitly flagged as non-linkable by setting the following boolean metadata to false:

```
node_parameters
{
    ...
    AiMetaDataSetBool(mds, "parameter_name", "linkable", FALSE);
}
```

Attempts to use AiNodeLink() on a parameter that has been flagged as non-linkable will produce a warning but otherwise have no effect. (trac#2341)

- **Application defined string:** Client applications, such as the Maya and Softimage plugins, are now allowed to pass a descriptive string to Arnold using the following API function:

```
AI_API AtVoid AiSetAppString(const char *appstr);
```

The app string will be printed in the output log and in the header of exported .ass files, which can greatly help developers when debugging customer scenes. We have also provided a corresponding Python binding. (trac#2337)

##### Incompatible changes

- **Renamed GI\_direct\_lighting to ignore\_direct\_lighting:** The ignore\_direct\_lighting option makes more sense alongside the other ignore\_\* options because it is mainly used for troubleshooting. Note that this flips the meaning of the option, so it is an incompatible change, not a simple rename. Along with the renaming, the -dd option in kick and pykick has also been changed to -idirect. (trac#2335)

##### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#1268	default auto-naming of nodes is not thread-safe	arnold	xo	major	2.33	2 years
#2344	Race condition in importance table initialization	arnold	xo	major	3.3	13 days
#2345	AiNode (node creation) function is not thread-safe	arnold	xo	major	3.3	12 days
#2346	Data race in string pool add	arnold	xo	major	3.3	12 days

Ticket	Summary	Keywords	Component	Owner	Priority	Milestone
#2355	SSS pointcloud build crashes on disjoint meshes	3.3.10.1	arnold	xo	major	3.3.11
#2359	Livelock in SSS pointcloud building	3.3.10.1	arnold	xo	major	3.3.11

## 3.3.10.0

### Milestone 3.3.10

#### Enhancements

- **Improved sampling of the skydome\_light with diffuse shading:** The MIS technique will now also be applied to the diffuse component of the standard shader when lit by skydome lights. This can result in a great reduction in noise in some cases, particularly for scenes lit by skydome lights that have low contrast maps applied. Note that this change also applies to all shaders that use the Oren-Nayar MIS API. (trac#2338)
- **Optimized skydome\_light and textured quad\_light importance sampling:** Shader color values for precomputing importance sampling are now stored alongside the importance table in a table of equal size. Using this table for lighting gives a dramatic speedup over re-evaluating the shader, particularly if the attached shader is expensive to evaluate. 2x and even 4x speedups are not uncommon. In most cases, the quality difference of this re-sampling is unnoticeable, and diminishes as skydome\_light.resolution and quad\_light.resolution increases. This feature can be disabled by setting enable\_fast\_importance\_tables to off. (trac#2342)

#### API additions

- **Metadata flag for non-linkable parameters:** Node parameters can be explicitly flagged as non-linkable by setting the following boolean metadata to false:

```
node_parameters
{
    ...
    AiMetaDataSetBool(mds, "parameter_name", "linkable", FALSE);
}
```

Attempts to use AiNodeLink() on a parameter that has been flagged as non-linkable will produce a warning but otherwise have no effect. (trac#2341)

- **Application defined string:** Client applications, such as the Maya and Softimage plugins, are now allowed to pass a descriptive string to Arnold using the following API function:

```
AI_API AtVoid AiSetAppString(const char *appstr);
```

The app string will be printed in the output log and in the header of exported .ass files, which can greatly help developers when debugging customer scenes. We have also provided a corresponding Python binding. (trac#2337)

#### Incompatible changes

- **Renamed GI\_direct\_lighting to ignore\_direct\_lighting:** The ignore\_direct\_lighting option makes more sense alongside the other ignore\_\* options because it is mainly used for troubleshooting. Note that this flips the meaning of the option, so it is an incompatible change, not a simple rename. Along with the renaming, the -dd option in kick and pykick has also been changed to -idirect. (trac#2335)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#1268	default auto-naming of nodes is not thread-safe	arnold	xo	major	2.33	2 years
#2344	Race condition in importance table initialization	arnold	xo	major	3.3	27 hours
#2345	AiNode (node creation) function is not thread-safe	arnold	xo	major	3.3	6 hours
#2346	Data race in string pool add	arnold	xo	major	3.3	6 hours

### 3.3.9.0

#### Milestone 3.3.9

**IMPORTANT NOTE:** This release has some rather significant changes to the SSS pointcloud generation code. Apart from the performance improvements and new sampling patterns, the `sss_max_samples` parameter has been removed and there is a great risk of old scenes performing very slowly or consuming large amounts of RAM because of this, i.e. pointclouds suddenly generating tens of millions of points instead of 100k points (which was the default value of `sss_max_samples`). If you were to encounter this, it is advised to increase the `sss_sample_spacing` until the number of samples is similar to the previous `sss_max_samples` setting.

#### Enhancements

- **Faster abort during SSS pointcloud generation:** The renderer will now poll for a render abort condition during the generation of the SSS pointcloud, which increases interactivity when computing dense pointclouds. (trac#2315)
- **Multi-threaded SSS pointcloud generation:** The blue noise SSS point cloud generator has been multi-threaded at the object level. This is achieved by spatially dividing the object and allowing different threads to generate points over each slice of the object. At low sampling rates, output may be different compared to previous releases, but quality remains the same and the results are deterministic as well as independent of the number of threads. (trac#2211)
- **Midpoint-based SSS pointcloud distribution:** Added a new `sss_sample_distribution` parameter to objects that allows the user to choose between the current `blue_noise` (default) sample generation method and two new midpoint-based generation methods called `polygon_midpoint` and `triangle_midpoint` which place SSS pointcloud samples on the midpoints of either the mesh's polygons or triangles. The midpoint-based sample generation methods generate samples much more quickly than the classic blue noise method, but shading quality will be determined by the number of faces on the mesh. (trac#2214)
- **Pref-based SSS pointcloud distribution:** A mode called `blue_noise_Pref` has been added to the `sss_sample_distribution` enum that causes the SSS pointcloud generation code to use the varying user-data field called `Pref` (if it exists) instead of the polymesh's P vertices. This should result in fixed sampling patterns as long as the `Pref` data and mesh topology remain constant during animation. (trac#997)
- **Alpha composition for atmospherics:** Atmospheric shaders may now optionally use the `sg->out_opacity` field of the shader globals to indicate to the trace methods exactly how opaque the atmospheric effect is. This information, if available, is later used to compute the sample's opacity and alpha. (trac#2314)
- **Plugin symbols can be exported globally:** Shared libraries loaded via `AiLibraryLoad()` (which includes custom Arnold shaders and procedurals) can optionally export their symbols globally instead of locally. This global export capability works in Linux and OSX only, and it is triggered by renaming the shared library file from `xxx.so` to `xxx.sog`. (trac#2323)
- **-ibump option in kick:** Similar to how `-idisp` ignores displacement, we have added an `-ibump` option to the kick command-line, which directly sets the `ignore_bump` option in Arnold. (trac#2316)

#### API additions

- **`AiShaderGlobalsGetVertexNormals/UVs()`:** These new functions allow you to get the vertex normals and UV coordinates for the current triangle. See the Doxygen docs for more details. (trac#2331, trac#2329):

```
AI_API AtBoolean AiShaderGlobalsGetVertexNormals(const AtShaderGlobals *sg, AtInt key, AtVector n[3]);  
AI_API AtBoolean AiShaderGlobalsGetVertexUVs(const AtShaderGlobals *sg, AtPoint2 uv[3]);
```

#### Incompatible changes

- **Renamed `autobump` to `disp_autobump`:** The `autobump` parameter, which was found in all geometric primitives, has been moved to the polymesh node, which is the only geometric primitive that support displacement mapping anyway. In addition, it has been renamed to `disp_autobump`, to better reflect that this is a displacement-related feature, along with existing `disp_*` parameters. (trac#2312)
- **Removed `sss_max_samples`:** The polymesh.`sss_max_samples` parameter, which placed a hard limit on the number of SSS points per object, has been removed. Because we no longer limit the maximum number of points, older scenes that were overdialed with a low `sss_sample_spacing` may now take a long time to render, unless the number of points is readjusted to reasonable levels by increasing the spacing. A more detailed reasoning behind this change is as follows: when we first designed the SSS system, we thought it would be a good idea to have a safety valve to avoid memory explosion when assigning a subsurface material to a very large mesh. Instead, this parameter has caused a great deal of confusion with users wondering why the `sss_sample_spacing` has no effect sometimes, or by thinking they are setting the number of points directly (which they aren't when the spacing is non zero). (trac#2320)

#### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2234	account for OIIO texture cache memory usage in the render statistics	arnold	marcos	major	3.3	2 months
#2325	'-set' option in kick doesn't override node parameters when using negative values	kick	oscar	major	3.3	3 days
#2327	'-set' option in kick doesn't override 'options.AA_samples'	kick	oscar	major	3.3	3 days
#2328	autobump doesn't filter texture maps	arnold	marcos	major	3.3	3 days

### 3.3.8.0

#### Milestone 3.3.8

##### Enhancements

- **Multilayer support for driver\_exr:** The OpenEXR driver (driver\_exr) now supports multiple AOVs. The resulting file channels are named based on the AOV names with the channel names appended, which is compatible with Nuke. Note that writing more than one AOV containing an alpha channel (e.g. "RGBA RGBA" or "A FLOAT") to driver\_exr may cause a conflict in the resulting file, and it is not well-defined which AOV's alpha channel will be used for the file's "A" channel. It is recommended that each driver has *at most* one AOV containing an alpha channel. (trac#1963)
- **Speedups for sss\_faceset:** Meshes using the sss\_faceset feature to eliminate SSS sample computation on selected faces will now perform even more efficiently by skipping inactive faces completely during the sample generation phase. (trac#2297)
- **Increased bump mapping precision far from the origin:** The bump mapping code has been modified to increase precision which results in less banding/moire artifacts in surfaces that are very far away from the origin. (trac#1381)
- **New ignore\_bump render option:** Just like we have ignore\_displacement, there is now an ignore\_bump option which will deactivate bump mapping effects in bump2d, bump3d and autobump. (trac#2300)
- **dPdu and dPdv differentials for displacement:** Displacement shaders now receive an estimate of dPdu and dPdv. This can be useful for tangent-space displacement maps, for example. Previously these values were always zero. (trac#2304)
- **Reject unmapped textures:** A new option texture\_accept\_unmapped was added to complement texture\_accept\_untiled. If enabled, this option causes a fatal error if an unmapped texture is sampled. This is useful for applications and workflows which enforce using only mipmapped images, as unmapped and untextured textures cause major performance penalties. Texture tiling and mipmapping can be done in a preprocessing step with the bundled maketx tool. (trac#2299)
- **Upgraded OpenImageIO to 0.10:** The bundled OpenImageIO version was upgraded from 0.8.8 to 0.10. This increases the accuracy and quality of texture sampling, as well as increasing the number of image formats supported. All OpenImageIO format plugins are bundled except cineon, field3d, and jasper. Note that 0.10 has more conformant handling for unassociated ("unpremultiplied") alpha. (trac#2285)
- **Static linking on Linux:** On Linux, OpenImageIO and all of its dependencies are statically linked into libai.so. This binds OpenImageIO and its dependencies like Boost and OpenEXR to Arnold at compile time. All non-API symbols (including RLM symbols) are hidden in the Arnold library. Arnold won't load outside versions of dependent libraries, nor will applications be able to load Arnold's versions of those libraries. Please note that this change makes it impossible to link to Arnold's bundled OpenImageIO. (trac#2296)
- **Static linking on OS X:** On OS X, OpenImageIO's dependencies are statically linked into libOpenImageIO.dylib, which is loaded by libai.dylib. This binds OpenImageIO to its dependencies like Boost and OpenEXR at compile time. This limits library conflicts to only OpenImageIO, alleviating some possible issues with dynamic versions of its dependencies. (trac#2296)
- **Static linking on Windows:** On Windows, the Intel Math Library is now statically linked into libai.dll. This may help prevent conflicts with applications using other versions of the Intel C++ compiler. (trac#2296)
- **Retries when requesting licenses:** Arnold can now be configured to request a license several times until it successfully acquires one. This can help in situations where the server is saturated by many simultaneous license requests. Users can set/get the number of attempts and the delay (in milliseconds) between failed attempts by either using the C API (see next section below) or by using the environment variables ARNOLD\_LICENSE\_ATTEMPTS and ARNOLD\_LICENSE\_ATTEMPT\_DELAY. If set via the API functions, those values are used. If not, the environment variables are used. Otherwise, Arnold defaults to just one (1) attempt and no delay. (trac#2236)

##### API additions

- **License request attempts:** As mentioned above, the number and delay of license request attempts can now be configured via the following functions:

```
AI_API AtVoid    AiLicenseSetAttempts(int n);
AI_API AtUInt    AiLicenseGetAttempts();
AI_API AtVoid    AiLicenseSetAttemptDelay(int msecs);
AI_API AtUInt    AiLicenseGetAttemptDelay();
```

##### Incompatible changes

- **libopenimageio.so moved inside libai.so in Linux:** As mentioned above, this change makes it impossible to link to Arnold's bundled OpenImageIO. We no longer distribute libopenimageio.so with the Arnold package. A very small number of customers relied on this libopenimageio.so file as a way for their OIIO-enabled custom shaders/apps to stay in sync with our OIIO version and thus avoid versioning conflicts at link time (that could lead to crashes). Because the OIIO library used by Arnold is now safely hidden behind the walls of libai.so, there are no more link conflicts; users are free to use whatever version of OIIO as they wish. (trac#2296)
- **Handling of unassociated alpha:** OpenImageIO 0.10 has more conformant handling for unassociated ("unpremultiplied") alpha, which is the default for PNG (as per spec), and conditionally for TIFF (based on the ExtraSamples tag). This may cause unexpected results for tools which consider alpha and color channels to be independent. (trac#2285)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2287	Crash releasing resources when instancing procedurals with unresolved references	arnold	oscar	major	3.3	11 days
#2298	Camera rays are not normalized	arnold	marcos	major	3.3	7 days

### 3.3.7.0

#### Milestone 3.3.7

##### Enhancements

- **New cylindrical\_light:** There is a new type of light available to lighters through a new node called cylindrical\_light. Like their cylinder shape counterparts, these lights are defined with two endpoints and a radius. These new lights should fully support all of the features available to lights in Arnold like motion-blur and MIS, excepting volumetrics/atmospherics which should will be supported in a future version. (trac#2195)
- **MIS support in diffuse layer of standard shader:** The necessary functions for MIS to work with the Oren-Nayar BRDF used by the standard shader have been implemented. Apart from being used internally by this shader, these functions are also available to shader writers inai\_shader\_util.h, where the rest of the API-provided MIS functions are located. (trac#2251)
- **MIS support in distant\_light:** Distant lights now correctly implement the required functionality for MIS to work with this type of lights. This should allow for cleaner specular highlights in glossy reflections of distant lights with angle > 0. (trac#2164)
- **Optimized importance sampling table precomputation:** The precomputation of the importance sampling tables for quad\_light and skydome\_light is now multithreaded. Any rendering threads which sample a quad\_light or skydome\_light that has not completed this precomputation will aid in computing the table, rather than wait for one thread to compute the table (as was the previous behavior). (trac#2258, trac#2266) The precomputation was made deterministic under threading and more numerically stable (output may change slightly), and is now safeguarded against NaN texels. (trac#2292)
- **Added ray differentials for all camera types:** Non-perspective camera projections such as fisheye and cylindrical now set derivatives properly, which allows texture filtering, wireframe shading and bump mapping to work correctly for shaders seen through these types of cameras. (trac#1234)
- **improved SSS precomputation with motion blur:** Arnold now uses a global absolute time to perform pointcloud lighting calculations. Previously the first key in the mesh would be used, but this can lead to artifacts when using centered frame motion blur for example because the first key is the position of the model at the previous frame. Now the mesh is positioned according to the global option sss\_time which default to 0 both for backwards compatibility and more consistent behavior with centered frame motion blur (trac#2284)
- **Environment variable expansion in searchpaths:**  
The texture\_searchpath, procedural\_searchpath and shader\_searchpath options now support expansion of environment variables delimited by square brackets. This works both using the API and inside .ass files as shown below: (trac#1816)

```
AtNode *options = AiUniverseGetOptions();
...
AiNodeSetStr(options, "procedural_searchpath", "[MY_ENVAR_PATH]/to/somewhere");

options
{
    AA_samples 4
    GI_diffuse_samples 4
    ...
    procedural_searchpath "[MY_ENVAR_PATH]/to/somewhere"
}
```

- **pykick enhancements:** There are several enhancements in how pykick.py is used and deployed: (trac#2271)
  - pykick.py is now an executable Python script which can be invoked directly via a new symbolic link pykick located in the bin directory. The symbolic link is only available in Linux/OSX.
  - pykick.py now points to the location of the Arnold Python bindings, so the user is no longer required to include that location in the PYTHONPATH environment variable.
  - Since the Arnold Python bindings use the ctypes module (introduced in Python 2.5), pykick.py now checks the current version of the Python interpreter used.

##### API additions

- **AiColor():** Added new convenience function that takes a float value and creates a color with all components set to that value (trac#2268)

##### Incompatible changes

- **auto\_transparency\_probabilistic now enabled by default:** The default value has changed from OFF to ON. This could affect the look of scenes containing semi-transparent or min\_pixel\_width curve shapes. (trac#2276)

##### Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2255	kick should only close its window when pressing ESC	kick	xo	critical	3.3	3 weeks
#2191	Random crashes with points.min_pixel_width > 0 and 1-element array in points.radius	arnold	oscar	major	3.3	2 months
#2245	Intel compiler floating point inconsistencies in Windows	arnold	xo	major	3.3	4 weeks
#2247	Problem loading some .OBJ format files	arnold	angel	major	3.3	4 weeks
#2249	Intersection precision problems in far away point primitives	arnold	xo	major	3.3	3 weeks
#2254	Intersection precision problems in far away sphere primitives	arnold	xo	major	3.3	3 weeks
#2259	incorrect MIS variable computation in textured, importance-sampled quad lights	arnold	alan	major	3.3	2 weeks
#2262	'min_pixel_width' in 'points' primitive is not properly working	arnold	oscar	major	3.3	2 weeks
#2265	AiEvaluateLightSample() is modifying lighting shader globals	arnold	alan	major	3.3	13 days
#2270	'maketx' and 'pykick' are not executables in linux releases	arnold	oscar	major	3.3	8 days
#2277	Buckets have different intensities when rendering with MIS and subdivision	arnold	xo	major	3.3	7 days
#2280	Linux/OSX kick process doesn't properly exit after display window is closed	kick	oscar	major	3.3	6 days
#2284	sss precomputation should occur at a fixed time	arnold	marcos	major	3.3	3 days
#2292	Crash when using a textured importance sampled light with NaN texels	arnold	xo	major	3.3	26 hours
#2293	Crash when opening a non-existent '.ass' file	arnold	oscar	major	3.3	25 hours
#2243	Rename "focal_distance" to "focus_distance" in "fisheye_camera"	arnold	oscar	minor	3.3	4 weeks
#2267	Windows kick process doesn't exit after display window is closed	kick	xo	minor	3.3	10 days

### 3.3.6.0

#### Milestone 3.3.6

##### Enhancements

- **Adaptive specular roughness clamping:** Clamping all specular reflections with the same minimum roughness after the first bounce greatly reduces the amount of fireflies in the final image due to reflective caustics, but can cause what should be mirror-like glossy reflections to become visibly blurred. A new adaptive technique has been implemented that will keep these reflections sharp when seen through other glossy reflections. (trac#2201)
- **Glossy refraction in the standard shader:** A new parameter called refraction\_roughness has been added to the standard shader allowing it to simulate microfacet-based glossy refraction effects. This parameter ties in to the IOR parameter to control the effect, causing refraction through rough surfaces with high IOR to appear blurrier than smooth surfaces with low IOR. The number of samples is controlled by the new GI\_refraction\_samples render option. (trac#2016)
- **RPATH points to \$ORIGIN in Linux:** On Linux systems, the RPATH binary header in libai is now hardcoded to \$ORIGIN. This makes libai to always link first with libraries residing in the directory where libai itself is located, avoiding loading libraries with mismatched versions that may be reached through the LD\_LIBRARY\_PATH environment variable. It is now ensured that libai will be linking with the libraries deployed in the official Arnold package as both libai and its dependencies are located in the package directory tree. Therefore we encourage users *not* to relocate the Arnold binaries outside the <arnold\_root\_dir>/bin directory. (trac#2135)
- **OIIO headers and libraries:** We now distribute the OpenImageIO headers and Windows .lib file for the same exact version of OpenImageIO that Arnold is linking with. For shader writers and developers who need to link to OIIO in their own code, this makes it easier to always be in sync with Arnold and avoid versioning conflicts. (trac#2089)
- **License diagnostic mode in kick:** A new option, -licensecheck [<port>@<host>], has been added to kick. This allows you to check the status of the license server and to retrieve the number of "total" and "in-use" licenses. Below is a usage example. (trac#2144)

```
$ kick -licensecheck
Connecting to license server on 5053@localhost ... OK

product:      arnold
version:      303
expires:      permanent
total licenses: 1
inuse:        0
timeout:      240
minimum timeout: 120
```

##### API additions

- **AiNodeDestroy():** This new API function allows you to destroy a node, releasing all of its associated memory allocations, which can be used to remove a node from the scene (as long as a render is not in progress). (trac#2202)
- **AiMicrofacetBTDFIntegrate():** This new API function implements the glossy refraction effects now available in the standard shader. Please note that, although anisotropic refraction is not yet implemented, we decided to "future proof" this API and request UV tangent vectors and two separate roughness values; for now, just pass NULL as the tangent vectors and pass the same value for both roughness values. (trac#2016)

```
AI_API AtColor AiMicrofacetBTDFIntegrate(const AtVector *N, AtShaderGlobals *sg, const AtVector
*u, const AtVector *v, AtFloat rx, AtFloat ry, AtFloat eta_i, AtFloat eta_o, AtColor
transmittance);
```

- **Python binding for AiNodeClone():** When this API function was introduced, it missed the corresponding Python binding. (trac#2204)
- **Set/Get license server:** New API functions have been added in order to programmatically set and get the port and hostname where the license server is located. The hostname and port set by AiLicenseSetServer() overrides the use of the values configured in the environment variables ARNOLD\_LICENSE\_HOST and ARNOLD\_LICENSE\_PORT. (trac#2103)

```
AI_API AtBoolean AiLicenseSetServer(const char* host, unsigned int port);
AI_API AtBoolean AiLicenseGetServer(char* host, unsigned int& port);
```

- **License status query:** A new API function has been added to check the connection to the license server, check the RLM service (and the existence of available licenses in the server), and get information about all the installed/inuse licenses in the server. (trac#2200)

```
AI_API AtInt AiLicenseGetInfo(const char* host, unsigned int port, std::vector<AtLicenseInfo>&
licenses);
```

## Incompatible changes

- **Renamed focal\_distance to focus\_distance:** We have decided that it makes more sense to call this camera control focus\_distance since that more accurately describes its functionality. "Focal distance" was sort of a mix between "focal length" (which has a well defined meaning in optics) and "focus distance". Note that the old name is still supported, at the cost of a "deprecated synonym" warning, so strictly speaking this is not an incompatible change yet. (trac#2220)
- **Hidden internal symbols in linux:** We were previously exposing internal Arnold symbols in Linux, which could cause symbol collisions with other applications (e.g. Katana). We now only export symbols which are explicitly and properly marked in the public Arnold header files with theAI\_API prefix. (trac#1966)

## Bug fixes

Ticket	Summary	Component	Owner	Priority	Version	Created
#2199	specular roughness clamp is broken	arnold	alan	critical	3.3	4 weeks
#2194	specify which shader global vectors are unnormalized	arnold	alan	major	3.3	4 weeks
#2196	Crash on OSX 10.5 with 'kick -info'	arnold	oscar	major	3.3	4 weeks
#2203	Arnold Python bindings broken on MacOSX	arnold	oscar	major	3.3	2 weeks
#2205	Missing AiColorCorrupted for AtRGBA type	arnold	angel	major	3.3	2 weeks
#2207	crash when switching off/on `skip_license_check`	arnold	oscar	major	3.3	2 weeks
#2218	failed bucket writes should abort the render	arnold	marcos	major	3.3	9 days
#2219	displacement padding should not be ignored for deformation motion keys	arnold	marcos	major	3.3	9 days
#2225	Procedurals don't recognize .ASS and .OBJ uppercase file extensions	arnold	oscar	major	3.3	6 days
#2233	quad_light importance map memory is not reported correctly	arnold	marcos	major	3.3	4 days

## 3.3.5.0

### Milestone 3.3.5

#### Enhancements

- **Support for linear subdivision:** Added a new linear enumerated value in the subdiv\_type parameter of the polymesh node, which allows for polygonal subdivision without undergoing Catmull-Clark smoothing. This can be useful when displacing flat surfaces. (trac#2040)
- **New modes of UV subdivision:** Added linear and smooth enum values in the subdiv\_uv\_smoothing parameter of the polymesh node. The linear mode finally makes Arnold compatible with textures created with Z-Brush's linear UV subdivision format. (trac#2118)
- **Added shader\_searchpath option:** This is similar to the existing {procedural|texture}\_searchpath parameters of the options node. .ass files that reference custom shaders are now self-contained and portable, without the need to manually specify the search path with other methods such as kick -l <path>. (trac#2142)
- **Polymesh hit refinement:** Added a mode where Arnold will attempt to refine the intersection point between a ray and a polymesh. This might help rendering artifacts in scenes with very large scene extents. This is enabled by setting enable\_hit\_refinement on the options node. (trac#2165)
- **Opacity support in utility shader:** The default shader is now able to render transparent objects. This may be useful when benchmarking scenes with many partially opaque surfaces. This feature is controlled by a new float opacity parameter. This setting has no effect until the object is tagged as opaque=false. (trac#2132)
- **Removed excessive warnings from uninstalled nodes:** When loading .ass files exported with node types that have not been installed/cannot be found, a "node is not installed" warning will now be printed out only once, potentially eliminating a large amount of redundant warnings. (trac#2134)
- **Optional warnings for shaders returning NaNs:** The render option shader\_nan\_checks has been added to selectively check for NaN results from each shader evaluation which can be used to help debug shading trees. This option has a slight impact on performance and should be activated only when necessary. (trac#1855)
- **Added option -qres to kick:** This is a handy command to quickly reduce the rendering resolution (i.e. quarter the number of pixels) when for example debugging scenes. (trac#2174)
- **Improved licensing messages:** The logging of licensing messages (of info, warning, and error types) has been changed to improve brevity and clarity. In addition, temporary licenses now report the expiration date along with the remaining days for which the license is valid. (trac#2127, trac#2154)
- **License heartbeat/timeout:** Arnold now emits periodic "heartbeats" to the license server, which allows for licenses that were checked out by clients which did not properly release them to be freed by the server after a given timeout. *Users who would like to use this new TIMEOUT feature are encouraged to visit the [License server timeout](#) wiki for configuration guidelines.* (trac#2140, trac#2143)

#### API additions

- **Move metadata specification to a separate file:** We have implemented support for external metadata files, which are text files with a simple format. For more information about the file format and how it works, see the [Arnold Metadata files](#) wiki. (trac#1937)
- **Inlined C++ function versions of AtColor, AtRGBA and AtVector macros:** Many utility macros for colors and vectors have been changed to inlined C++ functions. This has several advantages over C macros (like referencing, type safety and return values) allowing client code, especially shaders, to become shorter and more readable. For example, we can turn this code:

```
AtVector T;
AiV3Normalize(T, sg->dPdv);
AtVector c;
AiV3Cross(c, a, b);
AtColor color;
AiColorLerp(color, sg->v, root_color, tip_color);
AtColor t;
AiColorClamp(t, transmittance, 0, 1);
```

Into this:

```
AtVector T = AiV3Normalize(sg->dPdv);
AtVector c = AiV3Cross(a, b);
AtColor color = AiColorLerp(sg->v, root_color, tip_color);
AtColor t = AiColorClamp(transmittance, 0, 1);
```

Note that we haven't removed anything from the API, we have only added new symbols. Existing code using the older C-style macros should compile without changes. However, the older macros are now deprecated and will eventually be removed in a major release. The arithmetic macros AiV3Add(), AiColorScale(), etc have also been deprecated in favor of the overloaded operators +, \*, etc which are much easier to use and result in much more readable code. (trac#2160)

- **Array operators for AtPoint and AtRGB:** Both colors and points/vectors now provide indexed access to their components using the [] operator. This can help simplify some algorithms where component access is determined on the fly. (trac#2187)

```

AtColor c;
c[0] = 1.f;           // equivalent to: c.r = 1.f;
float b = c[2];      // equivalent to: b = c.b;

AtPoint point;
p[0] = 1.f;           // equivalent to: p.x = 1.f;
float y = p[1];       // equivalent to: y = p.y;

```

## Bug fixes

Ticket	Summary	Owner	Priority	Version	Created	Modified
#2136	Ignore 'abort_on_license_fail' when 'skip_license_check' is enabled	angel	major	3.3	8 weeks	8 weeks
#2141	NaNs/Infs at low roughness settings in the standard shader	alan	major	3.3	7 weeks	4 weeks
#2146	Python binding for AiArray{Set Get}Mtx() crashes	oscar	major	3.3	6 weeks	3 days
#2149	framebuffer memory not accounted for in the statistics	marcos	major	3.3	5 weeks	5 weeks
#2161	NaNs in transmittance effects of the standard shader	alan	major	3.3	3 weeks	3 weeks
#2166	specular_anisotropy affecting roughness of isotropic BRDFs in the standard shader	alan	major	3.3	3 weeks	12 days
#2170	Crash when the color parameter of a quad light is mapped with MIS on and a volume_scattering atmosphere shader	alan	major	3.3	2 weeks	6 days
#2181	fully transparent surfaces at max auto transparency depth causing artifacts	alan	major	3.3	6 days	4 days
#2182	AiParseParameterString() not thread safe	alan	major	3.3	4 days	4 days

### 3.3.4.2

#### Milestone 3.3.4

This release is the result of a lot of hard work over the last few months and it represents our best Arnold ever. There are many important fixes including OSX support, the flatness subdiv mode works incredibly well now, UV smoothing matches Softimage, physically\_based is now enabled (most BRDF's were Pi times brighter than they should). Note that, although this is a binary-compatible release, the look of the images will change; please read these notes carefully and think twice before upgrading your shows.

#### Enhancements

- **Mac OSX support:** Beginning with this release, we will provide official binary packages for the Mac OSX 64-bit architecture (i.e. darwin\_x86\_64). This new platform is fully supported, including the license usage. Existing licenses can be used for Mac OSX clients. (trac#1923)
- **UV smoothing controls in Catmull-Clark subdivision:** Added a subdiv\_uv\_smoothing parameter to the polymesh object that allows the user to choose between the classic pin\_corners UV subdivision mode and the new pin\_borders mode that should prove a better match to the way some other software packages like Softimage handle UV coordinate subdivision. (trac#2102)
- **Improved pixel error heuristic in Catmull-Clark subdivision:** The estimation of pixel edge lengths was numerically unstable, which could result in "popping" from adaptive subdivision with certain camera moves. We have switched to a more robust estimate that reduces this problem. (trac#2121)
- **Improved adaptive Catmull-Clark subdivision rate estimates:** The estimate used to guess the iteration count for each patch of a subdivision mesh has been greatly improved, making the use of adaptive subdivision (i.e. subdiv\_pixel\_error > 0) more appealing than ever. In particular, the flatness heuristic is now much more accurate at detecting flat regions. This results in smoother results using fewer triangles in most cases. The new algorithm can use less *peak* memory, but most importantly it generates vastly better results, and so can easily generate less *retained* memory too, about 5x for the same quality. (trac#2124)
- **Importance sampling of textured quad lights:** When a shader is connected to a quad\_light, we automatically construct importance sampling tables similar to those used in the skydome\_light. This permits efficient sampling according to the luminance of the texture, which can greatly reduce sampling noise, specially when using HDRI textures. Like the skydome\_light, the resolution of the table is controlled by the resolution parameter with a default value of 512. (trac#2026)
- **Corrected BRDFs for energy conservation:** A new physically\_based render option has been added that causes the BRDF functions to preserve instead of generate energy under direct lighting. This resolves many of the divide-by-Pi issues that were previously occurring. Note that the default value for this parameter has been set to true which **WILL** change the lighting of scenes currently using the following affected BRDFs: (trac#2010)

```
AiOrenNayarBRDF()
AiLommelSeeligerBRDF()
AiModifiedPhongBRDF()
AiStretchedPhongBRDF()
AiOrenNayarIntegrate()
AiLommelSeeligerIntegrate()
AiDirectDiffuse()
```

- **User-selectable BRDFs in the standard shader:** We have added a specular\_brdf parameter to the standard shader. You can use it to select the BRDF used to compute the specular highlight and glossy reflections. Available BRDFs are stretched\_phong, cook\_torrance and ward\_duer. All these BRDFs support MIS for efficient sampling. (trac#2012,trac#2013)
- **standard shader's specular controls more easily mappable:** There is a new specular\_roughness parameter in the standard shader which is capable of the full range specular effects with an input parameter in the [0,1] range. This is much easier to drive through a texture map than the [0,+inf] parameter range used by the Phong\_exponent parameter. (trac#2015)
- **Anisotropy controls in the standard shader:** Two new controls called specular\_anisotropy and specular\_rotation have been added to the standard shader which permit anisotropic effects in the BRDFs that support them. (trac#2091)
- **Improved BRDF sampling at grazing angles:** The annoying bright dots at grazing angles in the stretched-Phong BRDF used by the standard shader have been eliminated. This was mostly evident in low-poly meshes, not so much in highly tessellated meshes. In addition, the Ward-Duer BRDF now has less edge darkening in low-poly meshes. (trac#2042, trac#2107)
- **Beer's Law in the standard shader:** Added a transmittance parameter to the standard shader that, following Beer's Law, allows the user to specify how well light is transmitted (and tinted) through the object's volume. This will affect not only refraction but also shadow rays (if the object is not marked opaque). (trac#2017)
- **Atmospherics affect shadow rays:** Paving the way for volumetric shadows and self-shadows, attenuation/absorption effects from atmospheric shaders will now affect shadow rays which may result in differences in the rendered image. (trac#2029)
- **Physically based volume scattering:** The built-in volume\_scattering shader has received a number of corrections to make it more physically plausible. The phase function has been normalized (dividing it by 4Pi if physically\_based is enabled), and the equation for forward/backward scattering has been corrected. Consequently, we have removed the phase\_function enum as it added little value. The scattering behavior is now completely controlled by the eccentricity parameter which has a [-1,+1] range. The default of 0 indicates isotropic scattering, positive values correspond to forward scattering, and negative values to backward scattering. (trac#2023)
- **Improved volume sampling:** The volumetric importance sampling code is now both simpler and more accurate, which drastically reduces volumetric noise near light sources. (trac#2101)
- **Added invert\_normals parameter to ambient\_occlusion shader:** When this mode is enabled, ambient occlusion rays are fired *in*

wards into the object. This can be used to simulate dirt and rust in corners. Of course, for this trick to work, you need a far\_clip distance that is smaller than the object width, or at least some distance falloff (otherwise all the rays are blocked and the result is pure black). (trac#2054)

- **Added ambocc shading mode in utility shader:** A new ambient occlusion mode has been added to the utility shader, which is the default Arnold shader. The maximum distance is hard-coded to 100, while the sampling control is taken from GI\_diffuse\_samples. (trac#2083)
- **Added u and v color modes in utility shader:** In addition to the uv mode, which shows both U and V coordinates at the same time, it is sometimes useful to visualize either the U or the V coordinate alone. (trac#2071)
- **UV coordinates in the sphere primitive:** The sphere primitive now has support for U and V coordinates, using the standard polar parametrization, with the poles in the Y axis. This allows it to be textured, which can be useful for shaderball previews, etc. Tangents are also defined in sg->dPdU and sg->dPdV. (trac#2077)
- **Sprite/billboard support in the points primitive:** A new quad mode has been added to the points.mode parameter which is perfect for rendering large amounts of sprites/billboards. Particles in quad mode have a natural UV parameterization which allows them to be texture-mapped. (trac#1956)
  - For quad mode, the size of the particles is controlled with points.radius, which represents the half-length (radius) in the U direction of the sprite particle.
  - There are two new parameters, which are only available when rendering in quad mode:
    - points.aspect: the aspect ratio of the sprite. length\_V = length\_U / aspect. The default value is 1, i.e. square sprites.
    - points.rotation: the counter-clockwise rotation angle, in degrees. It can be negative and greater than 360. The default value is 0.
- **Motion blur optimization:** Top-level acceleration structures now take into account deformation in certain cases, leading to large speedups for scenes with fast moving, deforming characters. (trac#2109)
- **Reduced memory usage for wide ray trees:** Very "wide" ray trees caused by aggressive sampling settings, or large amounts of internal reflections and refractions no longer cause memory spikes due to a restructuring of memory allocations inside the renderer. The memory usage is now proportional to the *depth* of the tree, not the number of rays per camera ray. (trac#2093)
- **Blackman-Harris pixel filter:** The new blackman\_harris\_filter node produces similar results to a gaussian filter but has smoother tails which reduces aliasing. The recommended width is 3. (trac#2094)
- **Log detailed information when a signal is caught:** When a signal is caught, a warning/error message is displayed besides a signal description. (trac#1921)
  - Signals reported as WARNING: SIGINT and SIGTERM.
  - Signals reported as ERROR: SIGHUP, SIGSEGV, SIGBUS, SIGFPE, SIGILL, SIGABRT and SIGQUIT.
  - The rest of signals are processed by the default signal handler.
- **Progress messages displayed in green:** The render-completed percentages are now displayed in green to make them easier to find in large log files. (trac#2110)
- **Added skip\_license\_check option:** The new global option skip\_license\_check disables license checking, assuming that the license is not available, which will make the renderer produce watermarked images. This can be handy when some client machines don't need a license, for example if a modeler or animator is using a lighter's workstation temporarily. Note that this could also be done by tinkering with the environment variables (ARNOLD\_LICENSE\_HOST, ARNOLD\_LICENSE\_PORT), but, apart from being more difficult, this could lead to a delay while the license is being checked and denied. For convenience, we also added a -sl command line option in kick. (trac#2095)
- **Upgraded OIIO to 0.8.8:** This OpenImageIO release includes several important bug fixes as well as minor improvements, most notably: (trac#2044, #2045)
  - ImageCache: check for max open files when re-opening a closed file, not just when opening for the first time, or we may exceed max\_open\_files, and thus possibly hit fundamental OS limits.
  - ImageCache: fix bug wherein an invalidated and modified file would continue to flush in subsequent invalidations, even if the file was not modified again.
  - Improved PNG write speed by 4x.

## API additions

- **New Ashikhmin-Shirley BRDF:** (trac#2011)
 

```
AtFloat AiAshikhminShirleyBRDF(const AtVector *L, const AtVector *V, const AtVector *N, const
AtVector *u, const AtVector *v, AtFloat nx, AtFloat ny);
AtColor AiAshikhminShirleyIntegrate(const AtVector *N, AtShaderGlobals *sg, const AtVector *u,
const AtVector *v, AtFloat nx, AtFloat ny);
```
- **MIS sampling methods for all BRDF's in the standard shader:** All of the specular BRDFs used by the standard shader have been extended to use MIS. This functionality is also available for shader writers to use via the following newly exported API functions: (trac#2013)

```

AiAshikhminShirleyMISBRDF()
    AiAshikhminShirleyMISPDF()
    AiAshikhminShirleyMISSample()
    AiAshikhminShirleyMISCreateData()
AiCookTorranceMISBRDF()
    AiCookTorranceMISPDF()
    AiCookTorranceMISSample()
    AiCookTorranceMISCreateData()
AiStretchedPhongMISBRDF()
    AiStretchedPhongMISPDF()
    AiStretchedPhongMISSample()
    AiStretchedPhongMISCreateData()
AiWardDuerMISBRDF()
    AiWardDuerMISPDF()
    AiWardDuerMISSample()
    AiWardDuerMISCreateData()

```

- **Derivative computation for skydome mapping functions:** The following versions of the `AiMapping*()` functions have been added that include derivative/tangent calculations: (trac#2028)

```

AiMappingMirroredBallDerivs()
AiMappingAngularMapDerivs()
AiMappingLatLongDerivs()
AiMappingCubicMapDerivs()

```

## Incompatible changes

- **BRDF and volume scattering normalization:** As mentioned above, several BRDF's have been corrected since they were  $\pi$  times brighter than intended. The volumetric scattering phase function has also been corrected since it was  $4\pi$  times brighter than intended. For backwards-compatibility, we have added an option to "roll back" these corrections; if you need to preserve the look in existing scenes that were prepared with a previous version (e.g. to finish an existing project), you can disable the new `physically_based` option, which is ON by default. We *strongly* recommend staying in physically-based correctness moving forward with new shows as this will produce better balanced shading overall. (trac#2010, trac#2023)
- **Adaptive subdivision has changed:** In adaptive flatness mode, `subdiv_pixel_error` is now vastly different. *ALL* old settings in existing scenes are worthless now, and there's no way (nor any point) to make a compatibility mode. However, the polygon count for existing scenes that use the default settings (regular non-adaptive subdivision, i.e. `pixel_error = 0`) will not change.
- **Quad lights are single sided:** The sidedness parameter on quad lights introduced too many complexities for volumetrics and textured quad support, while offering very few advantages. It has been removed. Quad lights are now always single-sided. To achieve double-sided quad lights, simply duplicate the light and flip it. (trac#2027)
- **Renamed KICK\_SEARCHPATH environment variable:** This relatively unknown kick-specific environment variable has been renamed to the more generic-sounding ARNOLD\_PLUGIN\_PATH, which we'll try to standardize on for external Arnold apps and plugins. Note that multiple paths can be specified, using the path separators native to each Operating System: (trac#2021)

Linux/OSX: /this/is/path/one:/this/is/path/two  
Windows: C:\this\is\path\one;D:\this\is\path\two

## Bug fixes

#2161	NaNs in transmittance effects of the standard shader		
#2141	NaNs/Infs at low roughness settings in the standard shader		
#2121	AiPixelLength is numerically unstable	6 weeks	6 weeks
#2112	Drivers use additional ~4GB when image/bucket size exceeds ~2^24 pixels	6 weeks	6 weeks
#2111	Shortcut syntax for single elem arrays is not working for constant user data	6 weeks	6 weeks
#2107	bright specks at grazing angles with the standard shader using the stretched-phong brdf	7 weeks	6 weeks
#2105	kick should not render when given an unknown command line option	7 weeks	7 weeks
#2090	Specular and Reflection Fresnel are broken in standard shader when Ks or Kr are connected to a shader network	2 months	2 months
#2088	fix bump mapping noise/precision issues	2 months	8 weeks
#2085	NaNs in volume scattering when integrating very short segments	2 months	2 months
#2084	modify ray->mindist to prevent rays from getting stuck between two nearly coincident surfaces	2 months	2 months
#2079	SSS computation doesn't obey affect_diffuse parameter from lights	2 months	6 weeks
#2076	Bad intersection test on disk points	2 months	2 months
#2074	Setting light.filters array to NULL causes a crash	2 months	2 months
#2061	lights that don't contribute to volumetrics should not be attenuated by atmosphere	3 months	3 months
#2059	AiRadiance and AiIrradiance are not thread safe	3 months	3 months
#2055	node_update is not called in the nodes of a procedural when load_at_init is set to false	3 months	2 months
#2051	Receiving a signal after AiEnd() can cause strange behavior	3 months	6 weeks
#2050	custom plugin filters crash with negative AA samples	4 months	3 months
#2049	Crash when a hair material has diffuse cache on and a SSS cache flush is done after a render	4 months	2 months
#2047	Linking a shader component (such as .r) discards the static value for the other components	4 months	3 months
#2045	Modified textures are being permanently invalidated	4 months	3 months
#2042	Standard shader produces bright dots at grazing angles	4 months	3 months
#2034	Problem loading .obj files with Windows-style EOLs	4 months	3 months
#2033	kick -info crashes with parameter names > 32 chars	4 months	3 months
#2032	Plugins in '.' are not loaded when "-l" option an/or plugin searchpath is specified	4 months	3 months
#2019	Writing instance parameters to .ass is broken	4 months	3 months
#2010	Correct BRDFs for energy conservation	4 months	2 months
#1935	kick overwrites the same image file with every frame (with -turn <n> option)	6 months	2 months
#922	kick doesn't override the output file with -o	3 years	2 months
<b>or</b>			

### 3.3.3.2

#### Milestone 3.3.3

##### Enhancements

- **Improved threading for SSS:** The multi-threading performance of the lazy SSS mode has been greatly improved; some users are reporting a 1.5-2x speedup in their SSS renders. Because the lazy mode is now always much more efficient than the non-lazy mode, we have decided to make it the default and eliminate the non-lazy codepath and the `sss_lazy_evaluation` option. (trac#1813)
- **Adaptive maximum recursion depth for curves:** The curves node will now adaptively select the recursion depth used for its intersection test in *ribbon* mode depending on the curvature of the segment. This can provide up to a 20% performance increase depending on the shape of the curves. Straighter curves will see a greater performance increase. There is no performance increase for the *thick* mode. *Note: This change may result in small differences when comparing to images rendered with previous versions, particularly at the tips of wide ribbons.* (trac#1917)
- **Probabilistic transparency for curves:** A new render option has been added called `auto_transparency_probabilistic` that activates an alternate method of calculating transparency for shaders supporting it that can result in many less rays being cast from semi-transparent surfaces. This works particularly well with the `curves.min_pixel_width` feature as well as with opacity-mapped curves. Experiments with curves indicate a substantial performance increase using this method at the cost of some additional grain that tends to disappear at a reasonable rate as the AA setting is increased. (trac#1968)

A new API function has been provided for those shaders that would benefit from the alternate transparency calculation method called `AiShaderGlobalsApplyOpacity()`. However at the time being, of the built-in shaders only the hair shader has been adapted to use this new transparency method. Here is a code example of how this can be used in a shader:

```
shader_evaluate
{
    // Evaluate opacity
    AtRGB opacity = AiShaderEvalParamRGB(p_opacity);

    // Early exit if the apply opacity function returns true.
    if (AiShaderGlobalsApplyOpacity(sg, opacity))
        return;

    // The rest of the shader code...
    ...
}
```

- **Optimized points primitive:** The points primitive now renders slightly faster. Additionally, non-motion blurred points submitted with motion keys are now simplified to only use a single key. (trac#1979)
- **Shading network support in volume\_scattering.density:** You can now attach an arbitrary shader network into the density parameter. This can be used to "fill" 3D space with non-homogeneous fog. For example, you can use a fractal 3D shader that shades based on `sg->P`. (trac#1875)
- **Extended roughness range in Ward BRDF's:** The roughness of the Ward and Ward-Duer BRDF's can now be set to values as low as `AI_EPSILON` ( $1.0e-4$ ). The previous lower limit was  $0.01$  ( $1.0e-2$ ), which was insufficient for very sharp anisotropic reflections. (trac#1915)
- **Option to disable internal reflections in standard shader:** There is a new parameter in the standard shader called `enable_internal_reflections`. Disabling this option avoids the exponential explosion of reflective+refractive rays that tends to happen in glass materials, resulting in a big speedup with only relatively minor differences in the look of the glass. This is currently set to TRUE to preserve the look in existing scenes but will change to FALSE in a future release. (trac#1964)
- **Support for loading .obj geometry files in procedurals:** Geometry in .obj format can now be loaded using the procedural node. There is support for triangles, arbitrary polygons, UV's, multiple shaders per mesh (shader is directly referenced from .obj material name) and compressed .obj files (files with .obj.gz extension). (trac#1865)
- **Ray and memory stats improvements:** We now report the memory used by loading plugins (i.e. dynamic libraries containing 3rd party shaders) and by shader message passing. In addition, we now report the number of lighting calculations: lightloops (cached, direct, indirect), indirect diffuse, indirect glossy. Finally, in the ray stats, the volume shadows are now reported separately from regular shadows. (trac#1991)
- **Consolidated frequent warnings:** There are a number of debug warnings that keep being ignored (because they are very frequent). Examples include passing identical deformation motion blur data to polymesh and curves nodes, or passing identity matrices. These warnings have been aggregated and we now emit "performance warnings" instead, so that they don't fill up the log files. (trac#2007)

##### API additions

- **`AiShaderGlobalsApplyOpacity()`:** This is used for probabilistic transparency. See above for more details including a usage example. (trac#1968)

## Incompatible changes

- **sss\_lazy\_evaluation on by default:** This feature is now always enabled so the option has been removed. (trac#1996, #1813)
- **Removed GI\_quickshade:** This feature is unused and has been deprecated for a while. All related parameters and options have been removed: (trac#2009)
  - polymesh.GI\_quickshade
  - options.GI\_quickshade\_dist
  - options.GI\_quickshade\_raymask

## Bug fixes

#1971	Memory leak when tracing rays in "free" render mode
#1978	crash in AI_RENDER_MODE_CAMERA mode using uninitialized shaderglobals struct
#2076	Bad intersection test on disk points
#2074	Setting light.filters array to NULL causes a crash
#2070	Remove several memory leaks found in testsuite
#2055	node_update is not called in the nodes of a procedural when load_at_init is set to false
#2049	Crash when a hair material has diffuse cache on and a SSS cache flush is done after a render
#1935	kick overwrites the same image file with every frame (with -turn <n> option)
#922	kick doesn't override the output file with -o
#2050	custom plugin filters crash with negative AA samples
#2047	Linking a shader component (such as .r) discards the static value for the other components
#2045	Modified textures are being permanently invalidated
#2042	Standard shader produces bright dots at grazing angles
#2034	Problem loading .obj files with Windows-style EOLs
#2033	kick -info crashes with parameter names > 32 chars
#2032	Plugins in '.' are not loaded when "-l" option an/or plugin searchpath is specified
#2019	Writing instance parameters to .ass is broken
#2006	memory leak in persp_camera
#2005	Potential leak/crash when changing bucket_scanning interactively
#2004	curves should never return a 0 length normal
#2002	Statistics can use the wrong number of threads
#1998	avoid false sharing in light LSD structs
#1995	increase texture_max_memory_MB to 512
#1994	Crash when using AiRadiance or Ailrradiance
#1992	memory pool calls realloc() with an invalid address
#1989	make it impossible to write AOVs for shadow rays
#1988	lower threshold when calculating surface derivatives
#1987	bug in regular patch classification during adaptive subdivision
#1981	Empty arrays not properly written to .ass files
#1977	ERROR   Inside box.c -- could not find box normal
#1976	valgrind warnings in patch subdivision
#1975	standard shader produces black/white dots in glossy reflections
#1974	motion blur artifacts in low primitive-count scenes
#1973	Crash on AiAOVSetRGB
#1972	darkening when MIS samples are increased
#1967	corrupt rays from AiCookTorranceIntegrate
#1944	output drivers don't allow extension different than image file format

### 3.3.2.2

#### Milestone 3.3.2

##### Enhancements

- **Ability to abort during subdivision:** There is now support for aborting during the subdivision process. This can be handy when working with big meshes and high subdiv settings which can take quite a while to subdivide and tessellate. (trac#1919)
- **skydome\_light:** The new skydome\_light uses importance sampling techniques for sampling an environment map. This can greatly reduce the amount of noise in Image Based Lighting (IBL) scenarios. (trac#1924, trac#1953) The new light has two unique control parameters.
  - **format:** This parameter is identical to the format parameter of the sky shader. Valid values are angular, mirrored\_ball,latlong and cubic. The default value is angular.
  - **resolution:** This parameter gives the user control over the resolution of the light's importance sampling tables. Increasing this setting for environment maps with relatively small and bright areas can help to reduce noise at the cost of some precomputation time and memory usage. However this value should not be bigger than the original resolution of the HDRI map.
- **MIS-support in standard shader:** The standard shader will now use Multiple Importance Sampling (MIS) when calculating its specular component for those lights who have their mis parameter activated. This can drastically reduce the noise of sampling large area lights from sharp specular highlights. MIS is also now active by default on all lights. (trac#1938, trac#1542, trac#1953)
- **Rendering after writing .ass files:** When the preserve\_scene\_data option is enabled, we can continue rendering after writing the scene to .ass, as no data is destroyed in the scene. Note that this option is intended for interactive/debugging sessions and should not be used when rendering final frames on the farm, at the risk of increased memory usage. (trac#1927)
- **Collapse arrays with 1 element on .ass:** When writing an .ass file, arrays with only one element will be collapsed to their shortcut syntax (trac#1909). So, instead of:

```
shader 1 1 NODE myshader
```

the output in the .ass file will be:

```
shader myshader
```

- **Upgraded OIIO to 0.8.7:** This release of OpenImageIO includes many improvements as well as fixes for a crash when TIFF contains certain XML strings, a fix for a deadlock related to the use of autotile and a fix for a massive slowdown when using automip with low cache sizes. It also adds support for 16-bit SGI images. (trac#1961)
- **Camera distortion with persp\_camera.uv\_remap:** Added support for distorting the camera rays based on a UV image linked to the new uv\_remap parameter in the persp\_camera. Camera rays will be warped according to the values in the (r, g) components. This has been tested in production with a program called Hype that generates such UV maps. (trac#1876)

##### API additions

- **AiLoadPlugin():** This new function allows loading a specific plugin library, instead of a whole directory. (trac#1918)
- **Python bindings for metadata iterator:** Added missing Python bindings for the metadata iterator API introduced in Arnold 3.3.1.0. (trac#1932)
- **Explicit symbol export:** We are changing the way symbols are exported in Linux. The AI\_DLL\_EXPORT macro will now explicitly make the symbol visible. This will not change anything yet, as we are globally exporting all symbols anyway. But, once we change the compiler settings to hide all symbols by default, only symbols with this macro will be exported. This change only affects the entry points for plugins (NodeLoader) and procedural nodes (Procedural) (trac#1962)

##### Incompatible changes

None.

##### Bug fixes

#1379	kick crashes with progressive rendering and -turn <n>
#1905	OIIO deadlocks
#1977	ERROR   Inside box.c -- could not find box normal
#1978	crash in AI_RENDER_MODE_CAMERA mode using uninitialized shaderglobals struct
#1976	valgrind warnings in patch subdivision
#1975	standard shader produces black/white dots in glossy reflections
#1974	motion blur artifacts in low primitive-count scenes
#1973	Crash on AiAOVSetRGB
#1972	darkening when MIS samples are increased
#1967	corrupt rays from AiCookTorranceIntegrate
#1971	Memory leak when tracing rays in "free" render mode
#1934	support for carriage returns in quoted string parameters
#1959	Crash when reading an AOV without first writing to it
#1958	massive slowdown with automip and files bigger than the cache size
#1957	Sidedness is not preserved for instances
#1954	receive_shadows should not disable self_shadows
#1953	MIS support in skydome_light
#1946	Incorrect size for oriented disk used when sampling spherical lights
#1945	Object matrix data is being destroyed after render
#1941	crash with 16-bit SGI texture maps
#1939	stale shader messages causing autobump artifacts
#1936	crash with certain Photoshop CS4-saved TIFF files
#1933	kick (progressive + turn) uses all outputs after the first turntable
#1930	NaNs when evaluating the Ward BRDFs

## 3.3.0.0

### Milestone 3.3

#### Enhancements

- **Hair diffuse cache:** Implemented a lazy caching scheme for diffuse illumination of hair, which can speed up hair-to-hair global illumination rendering by up to 3x with minimal impact on visual quality. The illumination is cached at spline control points. This lazy cache can be enabled or disabled using the new boolean parameter `diffuse_cache` in the built-in hair shader (conservatively disabled by default). For custom shaders, see the `AiHairDirectDiffuseCache()` API below. (trac#1857)
- **Better FOV control in the cyl\_camera node:** There are now separate controls for vertical and horizontal FOV in the `cyl_camera` node. The new parameters are `vertical_fov` and `horizontal_fov`. We have provided a backwards-compatibility synonym for the old `fov` parameter name, which maps to the new `horizontal_fov` parameter name. There is also a new projective parameter that controls the angular mapping in the vertical axis; when set to true, the vertical mapping is projective, resulting in a standard cylindrical camera projection, and when set to false the vertical mapping is linear, resulting in a lat-long camera projection. Note that the projective parameter is enabled by default, which is a change in behaviour, as previously we always used a hardcoded linear mapping in the vertical axis. (trac#1863)
- **Kick command line shortcuts:** Added handy kick shortcuts to set the default color mode (-cm) and default shade mode (-sm). The following two lines are equivalent: (trac#1859)

```
kick file.ass -set ai_default_reflection_shader.color_mode polywire  
kick file.ass -cm polywire
```

- **New framework for interactive node updates:** We have redesigned the API for nodes to better support interactive changes. This has solved a critical memory leak when rendering multiple frames. See a more detailed description of the API changes below. (trac#1861)

#### API additions

- **API for hair diffuse cache:** Custom hair shaders can call the following API function that returns a fast, cached value of the indirect diffuse component: (trac#1857)

```
AI_API AtColor AiHairDirectDiffuseCache(const struct AtShaderGlobals *sg);
```
- **Refactoring of node initialization:** We have modified the semantics of the `node_initialize` and `node_destroy` methods to avoid inconsistencies and ambiguous behavior. Also, we have added a new `node_update` method for interactive modification of nodes (trac#1861). The new semantics are:
  - `node_initialize` is only called once per render session. It will be called right before the first call to `AiRender()`.
  - `node_destroy` is only called once at the end of the render session, during the call to `AiEnd()`.
  - `node_update` is called once per rendered frame. It is called at the beginning of `AiRender()`.

*Example:*

```
node_initialize  
{  
    AtCustomData *data = (AtCustomData*)AiMalloc(sizeof(AtCustomData));  
    node->local_data = (AtVoid*)data;  
}  
  
node_update  
{  
    AtCustomData *data = (AtCustomData*)node->local_data;  
  
    // Evaluate data based on current scene state.  
    // This is called at the begining of each rendered frame  
    // (such as each "pass" of an interactive session with  
    // progressive refinement).  
}  
  
node_destroy  
{  
    AiFree(node->local_data);  
}
```

- **Support for separately linking shader components:** When creating links between nodes in a shader network, you can now specify a specific component (e.g. R, G, B) on the target parameter, so that the source shader will be linked only to that component. In addition, each component of a parameter can be linked separately, even to different shaders. (trac#1811)

*Example:*

```
noise
{
    name noisesource
}

flat
{
    name shader1
    color.r noisesource
}

flat
{
    name shader2
    color.b noisesource
    color.r noisesource
}
```

*At the same time, the AiNodeLink() and AiNodeGetLink() API functions have support for component specification in the parameter name:*

```
AiNodeLink(shader1, "color.r", noisesource);
```

*For now, only target parameters support separate linking of shader components. In the future we will also add support for specifying components in the source data (e.g. linking the red component of a node's output to the green component of a node's input parameter).*

- **AiStretchedPhongBRDF():** The rarely-used retro parameter is now optional and will have FALSE as its default value, if the parameter is not set. (trac#1793)
- **AiCellular():** The delta and ID parameters are now optional and will have NULL as their default value, if the parameters are not set. (trac#1867)

## Incompatible changes

- **AiCameraInitialize():** This function has been split in two. So, there is a new AiCameraUpdate() function, to be called from node\_update. (trac#1861) The signature for both is:

```
AtVoid AiCameraInitialize(AtNode *node, AtVoid *data);
AtVoid AiCameraUpdate(AtNode *node, AtBoolean plane_distance);
```

- **AiFilterInitialize():** This function has been split in two. So, there is a new AiFilterUpdate() function, to be called from node\_update. The signature for both is:

```
AtVoid AiFilterInitialize(AtNode* node, AtBoolean requires_depth, const char** required_aovs,
AtVoid* data);
AtVoid AiFilterUpdate(AtNode* node, AtFloat width);
```

- **node\_finish:** The signature for this method has been modified, removing the (often misused) destroy\_node parameter:

```
AtVoid node_finish(AtNode *node);
```

- **node->connectors:** The type of this member variable in the AtNode struct has changed from AtNode\* to AtConnectorData, to accomodate per-component linking. See ai\_nodes.h for details.
- **AiNodeLink():** Passing NULL in the source node as a way to "unlink" an existing node link is no longer supported, resulting in a warning message and the function doing nothing. You should instead use AiNodeUnlink(). (trac#1867)

## Bug fixes

#1854	-INF values in alpha channel are not properly filtered on the beauty
#1856	possible deadlock with SSS and instancing
#1864	AiNodeReset() should remove all shader links
#1858	light sample cache can allocate the wrong number of samples in SSS

### 3.3.1.0

#### Milestone 3.3.1

##### Enhancements

- **Subdiv optimizations:** The performance and temporary memory usage of subdivision surfaces has been improved by rewriting some internal data structures. Subdivision time is up to 4x faster on some scenes, while using up to 50% less peak memory. (trac#1888)
- **Faster writing of .ass.gz files:** The zlib compression settings are now optimized for speed instead of file size. The new .ass.gz files are slightly bigger (~5-10%) but are generated 3-4x faster. (trac#1889)
- **Faster Oren-Nayar BRDF evaluation:** Optimized the implementation of AiOrenNayarBRDF() by reducing the amount of expensive trigonometric function calls. (trac#1898)
- **Support for writing .ass files after render:** There is a new parameter in the global options called preserve\_scene\_data, disabled by default. If enabled, it will keep the original geometry data (normally removed after node initialization to save memory) and allow you to save the scene to .ass even after all the multiple re-renders and modifications that happen in an interactive session. This option is only intended for interactive modifications of scenes (e.g. light/shader tweaking), and not for final or batch rendering. It has a tradeoff, as the memory occupied by the scene will be higher when set. (trac#1892)
- **kick waiting for keypress:** There is a new -nokeypress command-line flag that makes kick return immediately after rendering instead of sitting there waiting for a keypress. (trac#1891)
- **Upgraded OIIO to 0.8.3:** This release of OpenImageIO includes many fixes like improved TIFF error messages, --checknan option for maketx, timer improvements and a clamping/wrapping fix for .png output files when alpha > color. (trac#1902)

##### API additions

- **New microfacet BRDF's:** Added two new physically-based BRDF's for shader-writers to use. These are modern microfacet-based BRDF's that both exhibit an off-specular peak and reduce the amount of edge darkening apparent in older, Phong-based BRDF's (like the stretched-Phong BRDF used in the standard shader).
  - Cook-Torrance microfacet BRDF (trac#1862): This is an implementation of the Cook-Torrance BRDF based on the Beckmann microfacet distribution. The new API is:

```
AI_API AtFloat AiCookTorranceBRDF(const AtVector *L, const AtVector *V, const AtVector *N,
                                     const AtVector *u, const AtVector *v, AtFloat rx, AtFloat ry);
AI_API AtColor AiCookTorranceIntegrate(const AtVector *N, AtShaderGlobals *sg, const
                                         AtVector *u, const AtVector *v, AtFloat rx, AtFloat ry);
```
  - Notes: This implementation omits the Fresnel term from the classic BRDF's equation, so it's up to the shader writer to add Fresnel effects to the shaders that use this BRDF. Also, although parametrized for anisotropy, this BRDF is currently isotropic so the u, v and ry parameters are ignored for the time being.
  - Improved Ward-Dür microfacet anisotropic BRDF (trac#1893): This is an implementation of the Ward-Geisler-Moroder-Dür BRDF, which is an improvement upon the Ward-Dür BRDF, which is itself an improvement upon the classic Ward BRDF. The new API is:

```
AI_API AtFloat AiWardDuerBRDF(const AtVector *L, const AtVector *V, const AtVector *N,
                                 const AtVector *u, const AtVector *v, AtFloat rx, AtFloat ry);
AI_API AtColor AiWardDuerIntegrate(const AtVector *N, AtShaderGlobals *sg, const AtVector
                                    *u, const AtVector *v, AtFloat rx, AtFloat ry);
```
- **Metadata iterator:** The new AtMetaDatalterator can be used to traverse all metadata attached to a node or one of its parameters. The API is similar to other existing iterators, as can be seen in the following usage example: (trac#1887)

```
AtMetaDataIterator *iter = AiNodeEntryGetMetaDataIterator(node_entry, "some_parameter");
while (!AiMetaDatalteratorFinished(iter))
{
    AtMetaDataEntry *entry = AiMetaDatalteratorGetNext(iter);
    printf("%s\n", entry.name);
}
AiMetaDatalteratorDestroy(iter);
```

We can choose whether we want to iterate over the metadata attached to a specific parameter (as in the previous example) or pass NULL as the parameter string to get metadata attached to the node itself:

```
AtMetaDataIterator *iter = AiNodeEntryGetMetaDataIterator(node_entry); // optional second
argument defaults to NULL
while (!AiMetaDatalteratorFinished(iter))
...
```

- **AOV read API:** We have implemented a set of API functions to read data from an AOV, similar to the existing API to set data. The new API is AiaAOVGet\*. (trac#1907)

- **AI\_LOG\_COLOR:** Added flag AI\_LOG\_COLOR in ai\_msg.h to enable or disable the use of different colors for log messages based on severity. (trac#1877)

## Incompatible changes

- **Oren-Nayar roughness:** The roughness parameter in AiOrenNayarBRDF() and AiOrenNayarIntegrate() has been normalized to the 0-1 range, which makes it easier to tweak and texture-map. Previously the range was 0-Pi. (trac#1911)
- **Walter fixes for Ward BRDF:** As described in Bruce Walter's widely-known technical report, the sampling equations from Ward's original paper were incorrectly weighted uniformly in all directions, making the results from AiWardIntegrate() overly bright and not physically plausible. We have applied Walter's fixes, but note that these corrected equations will severely darken the results from AiWardIntegrate() (especially at grazing angles) and it is recommended that shader writers use the newly introduced Improved Ward-D<sup>1</sup>r BRDF instead. (trac#1893)

## Bug fixes

#1899	crash in catclark code when face_visibility is false on all faces
#1814	Add signal handling in Linux to do scene cleanup on termination
#1903	memory usage stats overflowing on Windows x64
#1913	PNG driver adds noise on fully saturated colors
#1912	flickering in SSS in static geometry
#1906	Missing operator function implementation in AtVector
#1904	problems piping commands to kick on Linux
#1901	Light nodes causing a crash when scene is aborted after initialization
#1900	print full backtrace when crashing in multiple threads
#1896	Bug in spot lights MIS
#1894	AiNodeSetRGBA() is not honoring the always_linear metadata
#1890	crash in AiNodeSetRGB/RGBA when passed a NULL node
#1884	reject attempts to declare varying user-data on instances
#1883	Bad ray intersection in quad light MIS
#1881	light sample cache should flush when traceset changes
#1880	light sample cache should check sg->skip_shadow
#1879	autobump should operate in the Displacement context
#1878	potential flickering in SSS at certain sample settings
#1874	abort and display message bank contents when full
#1873	crash with a fatal message when nans are encountered in displacement
#1872	off-by-one error in hair caching code
#1870	hermite basis matrix in the curves primitive is incorrect

## 3.2.5.0

### Milestone 3.2.5

#### Enhancements

- **Implicit AOV type conversion:** You can now set AOV values with a different (but compatible) type than the type used to declare the AOV output. The values will be implicitly converted, following the same implicit conversion rules already used for shader network connections. For example, you can use AiAOVSetRGB() on an AOV that was declared as RGBA, which will result in an implicit alpha of 1.0. Previously this resulted in empty images that were puzzling for first time users. (trac#1831)
- **Ignore default values when writing to .ass:** This makes for a nice reduction in size for written .ass files. (trac#1840)
- **AOV sample clamping:** Sample clamping was already available for the main RGB (beauty) buffer, via options\_AA\_sample\_clamp. This has now been extended to work for any generic AOV's of RGB/RGBA type, and can be enabled with the newoptions\_AA\_sample\_clamp\_aovs. This is disabled by default. (trac#1848)
- **Exposure control in light sources:** We have added an exposure parameter to all light sources, which gives additional control over light intensity. The final light intensity is calculated according to the formula  $\text{intensity} * 2^{\text{exposure}}$ . The default value is 0 for backwards compatibility. (trac#1851)
- **Added console colors to log messages:** Console log output will use ANSI color codes to mark warnings in yellow and errors in red, for easier debugging. Not available in Windows. (trac#1849)

#### API additions

- **Comparison operators for AtPoint2 and AtRGBA:** The == and != operators were missing. Note that as these operators were already implemented for AtRGB and AtVector. (trac#1844)

#### Bug fixes

#1850	NaN's in hair with volumetrics and min_pixel_width > 0
#1564	artifacts in volume importance sampling with linear light falloff
#1852	Filter NaN values in generic AOVs
#1845	allow motion-blurred spheres to have constant radius
#1843	Fix OIIO texture cache flush
#1842	Comparison operators in AtVector and AtRGB should be const
#1841	Multiple SSS lookups not working properly with sss_subpixel_cache
#1839	AiNodeSetPtr does not work on user data of type AI_TYPE_NODE
#1838	Opening an .ass.gz file from a procedural doesn't work

## 3.2.4.0

### Milestone 3.2.4

#### Enhancements

- **IPR matrix update:** We have added support for matrix modifications during an IPR session. Changes will be instantly reflected in the render, without the need for destroying the scene (i.e. going to AiEnd() and recreating the scene from scratch). This works for all basic geometric nodes such as polymesh and curves, but also for more complex nodes like ginstance and procedural. (trac#1828)
- **Double-precision triangle intersection:** We have added support for a more robust ray-triangle intersector in the polymesh node. Because the new intersector uses double-precision math, it is slightly slower than the default intersector. This can be useful in extreme cases where holes are visible in polygon edges. The new intersector is enabled by the double\_precision\_intersector parameter in the polymesh, as shown in the example below. (trac#1834)

```
polymesh
{
    name my_mesh
    double_precision_intersector true
    ...
}
```

- **Upgraded OIIO to 0.8.2:** We have updated Arnold's OpenImageIO library dependency to release 0.8.2. This includes important improvements in the performance of the maketx tool. (trac#1836)

#### Bug fixes

#1832	Background shaders are not visible unless they set opacity to != BLACK
#1835	validate procedural output

### 3.2.3.1

#### Milestone 3.2.3

##### Enhancements

- **EXR layer names changed to R,G,B:** Instead of prepending the AOV name to the layer name, we now use R, G, B, A for the layer names. This can be controlled with the preserve\_layer\_name option in the driver\_exr node. (trac#1806)
- **Multi-path support in procedural\_searchpath and AiLoadPlugins():** You can now specify multiple search paths separated by the ":" character in Linux/OSX or the ";" character in Windows. For example: "some/path:another/path:yet/another/path". (trac#1807, trac#1817)
- **Reduced noise in glossy inter-reflections in standard shader:** We now clamp the Phong exponent to a maximum value of 50 for secondary rays, greatly reducing spike noise in sharp glossy reflections seen through rough glossy reflections. (trac#1818)
- **Support for polygons with more than 255 points:** The maximum number of points per polygon in the polymesh node has been increased from 255 to 65535. (trac#22)

##### API additions

- **AtParamIterator:** This API lets you iterate over the built-in parameters of a node. This was already possible by using the AiNodeEntryGetNumParams() and AiNodeEntryGetParameter() functions, which are now deprecated. The example below illustrates the recommended usage. (trac#1810)

```
AtParamIterator* iter = AiNodeEntryGetParamIterator(node_entry);
while (!AiParamIteratorFinished(iter))
{
    const AtParamEntry* pentry = AiParamIteratorGetNext(iter);
    printf("Built-in parameter : %s\n", AiParamGetName(pentry));
}
AiParamIteratorDestroy(iter);
```

- **AtUserParamIterator:** This API lets you iterate over the user-defined parameters of a node. Previously there was no way to do this. See the example below. (trac#1810)

```
AtUserParamIterator* iter = AiNode GetUserParamIterator(node);
while (!AiUserParamIteratorFinished(iter))
{
    const AtUserParamEntry* upentry = AiUserParamIteratorGetNext(iter);
    printf("User parameter : %s\n", AiUserParamGetName(upentry));
}
AiUserParamIteratorDestroy(iter);
```

##### Bug fixes

#1582	untiled texture maps are redundantly loaded once per thread
#1820	Texture access to the wrong texture (hash collision issues)
#1827	Avoid loading sitoa plugin from AiLoadPlugins
#1826	Upgrade OpenImageIO to 0.8.1
#1825	quad area lights with inverted sidedness are broken
#1824	kick doesn't use gamma correction on display driver
#1823	kick command line options should not depend on their order
#1822	quote all string parameters when writing to .ass
#1821	memory leak when removing polymesh invisible faces with face_visibility

## 3.2.2.0

### Milestone 3.2.2

#### Enhancements

- **Reflection/refraction exit colors in standard shader:** It is now possible to specify an exit color other than black for reflection or refraction rays that reach their respective maximum depth. Four new parameters have been added: (trac#1801)

```
% kick -info standard | grep exit
RGB      reflection_exit_color      0, 0, 0
BOOL     reflection_exit_use_environment false
RGB      refraction_exit_color      0, 0, 0
BOOL    refraction_exit_use_environment false
```

- **Configurable AOV names in standard shader:** You can now change the AOV name strings in case the default values are not appropriate. This is controlled by eight new string parameters: (trac#1804)

```
% kick -info standard | grep aov
STRING   aov_direct_diffuse      direct_diffuse
STRING   aov_direct_specular    direct_specular
STRING   aov_emission          emission
STRING   aov_indirect_diffuse    indirect_diffuse
STRING   aov_indirect_specular  indirect_specular
STRING   aov_reflection         reflection
STRING   aov_refraction        refraction
STRING   aov_sss                sss
```

#### Bug fixes

#1802	standard shader: remap range of diffuse_roughness to [0,1]
#1803	gobo rotation should be motion-blurred

### 3.2.1.0

#### Milestone 3.2.1

##### Enhancements

- **AOV support in standard shader:** The standard shader now writes to eight different AOVs: (trac#1791)

```
emission  
direct_diffuse  
direct_specular  
indirect_diffuse  
indirect_specular  
reflection  
refraction  
sss
```

- **Better AOV type checking:** You will now get a warning message if you try to write to an AOV specifying the wrong output type: (trac#1792)

```
00:00:00    6mb      |  starting 2 bucket workers of size 64x64 ...  
00:00:00    7mb WARNING |  [aov] type mismatch for "direct_diffuse" (RGB vs RGBA)  
00:00:02    7mb      |  bucket workers done in 0:02.05
```

- **Added standard.roughness parameter:** The diffuse layer in the standard shader uses a Lambert BRDF when roughness is zero (the default value) and an Oren-Nayar BRDF when roughness is greater than zero. The range is from 0 to Pi. (trac#1796)
- **New utility shader mode bad\_uvs:** This new mode helps when debugging problematic geometry, by clearly marking polygons where UV mapping is broken (zero or non-finite derivatives). Problems in the U coordinate are shown as red polygons, and problems in the V coordinate are shown as blue. (trac#1800)

##### Incompatible changes

- **Removed legacy catclark\_subdiv node:** This deprecated node existed solely for backwards compatibility with old scenes in our development testsuite. The polymesh.subdiv parameter, which accepted a catclark\_subdiv node, has also been removed. (trac#1790)
- **Removed curves.self\_intersectable parameter:** This fixes the ugly self-shadowing artifacts that you would get in ribbon mode with the default settings. (trac#1798)
- **Removed min() and max() Python methods:** These two AtPoint methods in the Python bindings were causing problems because they were aliasing the built-in Python functions. (trac#1799)

##### Bug fixes

#	Description
#1794	crash in AiCellular() when passing NULL delta/ID
#1795	standard shader: SSS layer should be affected by Fresnel term
#1789	remove dither_amplitude warnings with kick's drivers
#1797	crash in AiNodeSetArray() when setting a parameter twice with the same array

## 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 succinct 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, maintainability and, in some cases, resulting in better performance (such as in cases where the equivalent C macro would result in redundant evaluations 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

### 3.1.1.0

#### Milestone 3.1.1

##### Enhancements

- **Indirect irradiance caching for shader networks:** Multiple calls to AiIndirectDiffuse() within a shader network will now be optimized so that the result is only calculated once and then reused. (trac#1307)
- **Smaller .ass files:** Vertex lists and other arrays of floats are now written to .ass with 8 decimal digits instead of 16. The size of a typical .ass file is almost cut in half, with no loss of precision. (trac#1684)
- **Generic display driver:** A new node called driver\_display is provided for the convenience of client applications. This driver has a callback mechanism that can be used to implement visualization of Arnold rendering for any target platform (Maya, XSI, OpenGL, etc). Because this driver encapsulates a lot of the "boiler plate" code needed for visualization drivers, it is generally much easier to use this rather than writing a custom display driver from scratch every time. It is also much cleaner to use this driver when the host application is written in a language other than C/C++ (such as Python). (trac#1702)

##### API additions

- **Python bindings:** Python bindings are now provided for the majority of the Arnold API. They provide a one-to-one mapping from Python to the Arnold C API, keeping the same names for types and functions. The kick tool has been rewritten in Python using these bindings (trac#1671), and is included in the Arnold distribution as an example on how to use them. See the README.python file for more information on how to use these bindings. Here is the minimal application to render an .ass file: (trac#1655)

```
from arnold import *

AiBegin()
AiASSLoad("test.ass")
AiRender(AI_RENDER_MODE_CAMERA)
AiEnd()

• Scene node iterator: These new functions allow iterating over all or a subset of the scene nodes. (trac#1703) Here is a usage example:

AtNodeIterator *iter = AiUniverseGetNodeIterator(AI_NODE_ALL);
while (!AiNodeIteratorFinished(iter))
{
    AtNode *node = AiNodeIteratorGetNext(iter);
    // do something with the node here ...
}
AiNodeIteratorDestroy(iter);
```

- **AiMsgUtilGetUsedMemory(), AiMsgUtilGetElapsedTime():** These functions provide access to elapsed frame time and used memory, so that custom log callbacks can have the same memory and timestamp information as the default log output. (trac#1698)

##### Bug fixes

1656	potential crash in BVH traversal
1669	AiMsgError is incorrectly limited by max warnings setting
1695	remove 'faces' stats in triangle tessellation
1707	TIR rays should be tagged as AI_RAY_REFRACTED
1700	arnold scene rebuilding can crash with very simple hierarchies
1685	spatial correlation in SSS pointcloud sampler
1684	float vertices written to .ass with too many digits
1679	user-data of type 'byte' doesn't work
1672	Arnold reports the wrong memory size for systems with more than 4GB in Windows

# Archive / Updates

Arnoldpedia is also downloadable as a PDF.

## PDF Archive

File	Modified
 A5ARP-200617-1210-147852.pdf	Jun 20, 2017 by Lee Griggs

```
<a href="https://support.solidangle.com/plugins/servlet/gadgets/ifr?container=atlassian&mid=2362243088384&country=GB&lang=en&view=default&view-params=%7B%22writable%22%3A%22false%22%7D&st=atlassian%3AUnAnHoEaOIPKqnOaWhyBqbgXtZAzO7E55Xu5L7wCXzL3sBvOIzukK1GMKfhl5ChOX8%2Butvl22QP9aR7ngk%2BwekiBHvHzO39eyGV6WrJGgN3PPzOP92W6716YS1qbW8D3Urzctg96ceDJKV3iLObUS5AfQIFVvpGY8DGrDvXMVNw0jriXFtoohBJmuWbCA2Mma%2BGqjOy5So695W9NnsmjUSZXQ7Y%2BF6yq8ebQcCMGGvtHrGveNfI38YC%2Ba%2BeGrZ3Ev%2FqjG7YkjP8L3Ark4zzBsElgsVnFdKVm%2Fmg4xBoHVFkLLmwvYZqVVwgN7al4ivlkEIN6Ry4K4XMc0BcPzYrXHsX8WH9j5qu3YOaKv4p1KSXIIMhvLOrclwQb0VzR3mMJEC2Lnw%3D%3D&amp;up_isConfigured=true&amp;up_isReallyConfigured=true&amp;up_title=Arnoldpedia%2520Updates&amp;up_titleRequired=true&amp;up_numofentries=20&amp;up_refresh=15&amp;up_maxProviderLabelCharacters=50&amp;up_rules=%257B%2522providers%2522%253A%255B%257B%2522provider%2522%253A%2522streams%2522%252C%2522rules%2522%253A%255B%257B%2522rule%2522%253A%2522key%2522%252C%2522operator%2522%253A%2522is%2522%252C%2522value%2522%253A%255B%2522A5ARP%2522%255D%252C%2522type%2522%253A%2522select%2522%257D%255D%257D%255D%257D&amp;up_renderingContext=&amp;up_keys=&amp;up_itemKeys=&amp;up_username=&amp;url=https%3A%2F%2Fsupport.solidangle.com%2Frest%2Fgadgets%2F1.0%2Fg%2Fcom.atlassian.streams.confluence%3Aactivitystream-gadget%2Fgadgets%2Fconf-activitystream-gadget.xml&amp;libs=auth-refresh#rpctoken=1180247118">Activity Stream</a>
```