API FAQ

What is the difference between AiNodeGet* and AiShaderEvalParam*?

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 Ai NodeGet* 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?

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 AiNo deGet* 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*?

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*?

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?

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 thenode_initialize or node_update callbacks, which are executed once per session or per IPR pass, respectively, rather than per shading sample.

Using Anisotropic Speculars

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 pass 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 UVcoordinatesareprovidedthisvectorwillbezero, 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?

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 andsidenessaresetto255 (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?

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?

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?

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?

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?

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?

To add a user-definedparameterinan.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,istheuser-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?

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?

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

- Setyouroutputtobetiledexr, zip is fine.
- Set the bucket scanning method to 'top'.
- Use a viewer like imf_disp.

What parameters support motion blur keys?

Geometry:

- For all shape nodes: matrix.
- Forthepolymeshnode:vlistandnlist.
- 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?

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. Onpolymeshnodes with subdiv_type settononeandsubdiv_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?

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?

`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

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.