Arnold 5 Porting Guide

Last updated: 13-Mar-17, 12:06

Types

Some types have disappeared:

```
AtUInt32 -> uint32_t
AtUInt64 ->
AtByte   -> uint8_t
```

Metadata

Replace metadata store argument with node entry in **AiMetaDataSetX()**:

```
AiMetaDataSetBool(mds, "input", "always_linear", true)    // arnold 4
AiMetaDataSetBool(nentry, "input", "always_linear", true) // arnold 5
```

AtParamValue

AtParamValue is no longer a union but a class, the members become accessors:

```
sg->out.RGB -> sg->out.RGB()
```

Note: Do not replace in the Python code, where members are accessed as if they were still a union.

Parameters

When declaring parameters, the deprecated **AiParameterXXX()** forms have been removed, replace with **AiParameterXxx()**. Also, the **PNT** and **PNT2** forms have to be replaced with **Vec** and **Vec2**:

```
AiParameterBYTE(n,c)    -> AiParameterByte(n,c)
AiParameterINT(n,c)     -> AiParameterInt(n,c)
AiParameterUINT(n,c)    -> AiParameterUInt(n,c)
AiParameterBOOL(n,c)    -> AiParameterBool(n,c)
AiParameterFLT(n,c)     -> AiParameterFlt(n,c)
AiParameterVEC(n,x,y,z) -> AiParameterVec(n,x,y,z)
AiParameterPNT(n,x,y,z) -> AiParameterVec(n,x,y,z)
AiParameterPNT2(n,x,y)  -> AiParameterVec2(n,x,y)
AiParameterSTR(n,c)     -> AiParameterStr(n,c)
AiParameterPTR(n,c)     -> AiParameterPtr(n,c)
AiParameterNODE(n,c)    -> AiParameterNode(n,c)
AiParameterARRAY(n,c)   -> AiParameterArray(n,c)
AiParameterMTX(n,c)     -> AiParameterMtx(n,c)
AiParameterENUM(n,c,e)  -> AiParameterEnum(n,c,e)
```

AtColor

The redundant **AtColor** type has been replaced with **AtRGB**:

```
AtColor -> AtRGB
```

Replace **AiColor()** with **AtRGB** constructor:

```
AiColor(r, g, b) -> AtRGB(r, g, b)
```

Replace **AtColorCreate()** with simple **AtRGB** member assignments:

```
AtColorCreate(rgb, r, g, b) -> rgb.r = r; rgb.g = g; rgb.b = b;
```

Alternatively, assign the an **AtRGB** allocated on the stack:

```
AtColorCreate(rgb, r, g, b) -> rgb = AtRGB(r, g, b)
```

Similarly with **AiRGBACreate()**, with **AtRGBA** member assignments:

```
AiRGBACreate(rgba, r, g, b, a) -> rgba.r = r; rgba.g = g; rgba.b = b; rgba.a = a;
```

Or with the **AtRGBA** constructor:

```
AiRGBACreate(rgba, r, g, b, a) -> rgba = AtRGBA(r, g, b, a)
```

Replace **AiColorLerp()** and **AiRGBALerp()** with the templated **LERP()**:

```
AiColorLerp(t, rgb1, rgb2)  -> L(t, rgb1, rgb2)
AiRGBALerp(t, rgba1, rgba2) -> L(t, rgba1, rgba2)
```

Replace **AiColorClamp()** with the templated **AiRGBClamp()**:

```
out = AiColorClamp(rgb, min, max) -> out = AiRGBClamp(rgb, min, max)
```

The obsolete define **AiColorIsZero()** must be replaced with **AiColorIsSmall()**:

```
AiColorIsZero(rgb) -> AiColorIsSmall(rgb)
```

Unless what you really want to test is if the color is pure black:

```
AiColorIsZero(rgb) -> (rgb == AI_RGB_BLACK)
```

Replace **AiColorEqual()** with a simple comparison:

```
AiColorEqual(rgb1, rgb2) -> (rgb1 == rgb2)
```

AI_RGBA_BLACK

To alleviate ambiguity, **AI_RGBA_BLACK** (0, 0, 0, 0) is now **AI_RGBA_ZERO**. Note that **AI_RGBA_BLACK** might come back as (0, 0, 0, 1) in subsequent versions of Arnold:

```
AI_RGBA_BLACK -> AI_RGBA_ZERO
```

Macros and utility functions

The **MIN()**, **MIN3()**, **MIN4()**, and corresponding **MAX** functions have all been renamed to **AiMin()** and **AiMax()**, which are overloaded and support 2, 3, or 4 arguments:

```
MAX(a, b)        -> AiMax(a, b)
MIN(a, b)        -> AiMin(a, b)
MIN3(a, b, c)    -> AiMin(a, b, c)
MIN4(a, b, c, d) -> AiMin(a, b, c, d)
```

Replace **ABS()** with **std::abs()** note that ai_math.h includes appropriate headers for both the floating point and integer versions of abs:

```
ABS(a) -> std::abs(a)
```

SGN() has disappeared:

```
SGN(a) -> (a < 0 ? -1 : 1)
```

AtVector

Replace **AiVector**() with **AtVector** constructor:

```
AiVector(x, y, z) -> AtVector(x, y, z)
```

The obsolete define **AiV3IsZero()** must be replaced with **AiV3IsSmall()**:

```
AiV3IsZero(vec) -> AiV3IsSmall(vec)
```

unless you really wanted to test if the vector was zero:

```
AiV3IsZero(vec) -> (vec == AI_V3_ZERO)
```

Replace **AiV3Create**() with **AtVector** assignment:

```
AiV3Create(vec, x, y, z) -> vec = AtVector(x, y, z);
```

AtPoint2

The **AtPoint2** type has been renamed **AtVector2**:

```
AtPoint2 -> AtVector2
```

The corresponding type ID is now **AI_TYPE_VECTOR2**:

```
AI_TYPE_POINT2 -> AI_TYPE_VECTOR2
```

All API functions containing **Pnt2** must be renamed with **Vec2** instead:

```
AiNodeSetPnt2()   -> AiNodeSetVec2()
AiArraySetPnt2()  -> AiArraySetVec2()
AiUDataGetPnt2()  -> AiUDataGetVec2()
AiParameterPnt2() -> AiParameterVec2()
[...]
```

The **AtParamValue** accessor **PNT2()** is renamed **VEC2()**:

```
sg->out.PNT2() -> sg->out.VEC2()
```

AtPoint

The **AtPoint** type is phased out and simply replaced with **AtVector**:

```
AtPoint -> AtVector
```

Thus **AI_TYPE_POINT** should be replaced with **AI_TYPE_VECTOR**. Often times, entire branches in **switch()** or **if()** branches will disappear in the **AI_TYPE_POINT** case. Watch out for **AI_TYPE_POINTER** occurences in search & replace operations:

```
AI_TYPE_POINT -> AI_TYPE_VECTOR
```

Replace **AtParamValue**'s **PNT()** accessors with **VEC()**:

```
sg->out.PNT() -> sg->out.VEC()
```

Replace **Pnt** suffix with **Vec** in all functions with type variants:

```
AiNodeGetPnt()   -> AiNodeGetVec()
AiArraySetPnt()  -> AiArraySetVec()
AiParameterPnt() -> AiParameterVec()
[...]
```

AtMatrix

AiM4Identity() no longer takes an argument but returns an **AtMatrix**:

```
AiM4Identity(mtx) -> mtx = AiM4Identity()
```

AiM4Transpose(), **AiM4Invert()**, **AiM4Mult()** lose the output argument and return an

AtMatrix:

```
AiM4Transpose(mtx, mout) -> mout = AiM4Transpose(mtx)
AiM4Invert(mtx, mout)    -> mout = AiM4Invert(mtx)
```

AiM4Frame(), **AiM4Translation()**, **AiM4Scaling()**, **AiM4Lerp()**, **AiM4RotationX()**, **AiM4RotationY()**, **AiM4RotationZ()** pointer arguments become references and they return an **AtMatrix**:

```
AiM4Frame(mout, &o, &u, &v, &w) -> mout = AiM4Frame(o, u, v, w)
AiM4Translation(mout, &t)       -> mout = AiM4Translation(t)
AiM4Scaling(mout, &s)           -> mout = AiM4Scaling(s)
AiM4RotationX(mout, rx)         -> mout = AiM4Rotation(rx)
```

AiM4VectorByMatrixMult(), **AiM4PointByMatrixMult()**, **AiM4HPointByMatrixMult()** and **AiM4VectorByMatrixTMult()** pointer arguments become references and they return an **AtVector**:

```
AiM4VectorByMatrixMult(&mout, mtx, &vec)  -> mout = AiM4VectorByMatrixMult(mtx, vec)
AiM4PointByMatrixMult(&mout, mtx, &vec)   -> mout = AiM4PointByMatrixMult(mtx, vec)
AiM4HPointByMatrixMult(&mout, mtx, &vec)  -> mout = AiM4HPointByMatrixMult(mtx, vec)
AiM4VectorByMatrixTMult(&mout, mtx, &vec) -> mout = AiM4VectorByMatrixTMult(mtx, vec)
```

AiM4Berp() has disappeared.

AtArray

Members are no longer accessible directly.

```
arr->nelements -> AiArrayGetNumElements(arr)
arr->nkeys     -> AiArrayGetNumKeys(arr)
arr->type      -> AiArrayGetType(arr)
```

Note: Verify Python API, these functions are not exposed.

User Data

AiUserGetX() functions take a reference instead of a pointer to the destination variable:

```
AiUDataGetVec2(uvname, &uv) -> AiUDataGetVec2(uvname, uv)
```

AiSampler

AiSampler() now takes a seed argument to avoid correlation artifacts when several samplers are present in the scene graph. In shaders, it is reasonable to use the node name hash as the seed:

```
static const uint32_t seed =
    static_cast<uint32_t>(AiNodeEntryGetNameAtString(AiNodeGetNodeEntry(node)).hash());
AtSampler* sampler = AiSampler(seed, nsamples, ndim);
```

AiBuildLocalFrameShirley

AiBuildLocalFrameShirley() has been replaced by **AiV3BuildLocalFrame()**:

```
AiBuildLocalFrameShirley(&u, &v, &N) -> AiV3BuildLocalFrame(u, v, N)
```

AiMakeRay

AiMakeRay() returns an **AtRay** and takes reference arguments, except optional **dir**, thus:

```
AtRay ray;
AiMakeRay(&ray, AI_RAY_REFLECTED, &sg->P, &dir, AI_BIG, sg); // arnold 4
```

becomes:

```
AtRay ray = AiMakeRay(AI_RAY_REFLECTED, sg->P, &dir, AI_BIG, sg); // arnold 5
```

AI_PROCEDURAL_NODE_EXPORT_METHODS(MyProceduralMtd);procedural_cleanup();//cleanuppro
cedural_get_node{} // GetNode(i) = MyProceduralMtd; node->output_type =AI_TYPE_NONE;
node->name = "my_procedural"; AI_NODE_SHAPE_PROCEDURAL; strcpy(node->version,
AI_VERSION); return true;

Output Driver Nodes

AiDriverInitialize() no longer has a data parameter, use **AiNodeSetLocalData**() instead:

```
AiDriverInitialize(node, supports_multiple_outputs, data); // arnold 4
```

becomes:

```
AiDriverInitialize(node, supports_multiple_outputs); // arnold 5
AiNodeSetLocalData(data);
```

AiDriverGetLocalData() is removed, use **AiNodeGetLocalData**() instead.

```
AiDriverGetLocalData() -> AiNodeGetLocalData()
```

There's no need to call **AiDriverDestroy()** anymore.

AtString

A lot of functions now require **AtString** arguments instead of **const char***.

AiState{Get|Set}MsgX(), **AiAOVSetX()** now take an **AtString** argument. For example:

```
AiStateSetMsgFlt("mylabel", f); // arnold 4
AiStateGetMsgFlt("mylabel", &f);
```

becomes:

```
static const AtString mylabel("mylabel"); // arnold 5
AiStateSetMsgFlt(mylabel, f);
AiStateGetMsgFlt(mylabel, &f);
```

Reciprocally however, functions expecting a **const char*** can be passed an **AtString** tranparently except with variadic arguments. In this case, you need to explicitely call the **AtString::c_str()** method:

```
AtString mystring("toto");
AiMsgWarning("my string: %s", mystring.c_str());
```

Light Sampling

AiLightsGetSample() returns the sample in an **AtLightSample** instead of storing it in the shader globals:

```
AiLightsGetSample(sg) -> AtLightSample ls;
                         AiLightsGetSample(sg, ls);
```

All light sample related shader globals are now mebers of **AtLightSample**. Note that **sg->we** has disappeared, replace it with **1/ls.pdf**:

```
sg->Lp    -> ls.Lp
sg->Ldist -> ls.Ldist
sg->Ld    -> ls.Ld
sg->Li    -> ls.Li
sg->Liu   -> ls.Liu
sg->Lo    -> ls.Lo
sg->we    -> 1.0f / ls.pdf
```

The functions **AiLightGetAffect{Diffuse|Specular}()** were removed, use directly **AiLightGet{Diffuse|Specular}()** instead:

```
if (AiLightGetAffectDiffuse(sg->Lp)) {…} -> if (AiLightGetDiffuse(ls.Lp) > 0.0f) {…}
```

Lighting functions

AiIndirectDiffuse() takes references instead of pointers and adds an **AtRGB** weight argument:

```
AiIndirectDiffuse(&sg->Nf, sg) -> AiIndirectDiffuse(sg->Nf, sg, AI_RGB_WHITE)
```

AiCookTorranceIntegrate() was been phased out, use **AiMicrofacetBSDF()** and **AiBSDFIntegrate()** instead. For example, this code:

```
indirect_specular = AiCookTorranceIntegrate(&sg->Nf, sg, NULL, NULL, specular_roughness,
                                            specular_roughness); // arnold 4
```

becomes:

```
AtRGB direct, indirect; // arnold 5
AtBSDF* bsdf = AiMicrofacetBSDF(sg, AI_RGB_WHITE, AI_MICROFACET_BECKMANN, sg->Nf, NULL,
                                0.0f, specular_roughness, specular_roughness);
AiBSDFIntegrate(sg, &direct, &indirect, bsdf);
indirect_specular = indirect;
```

Shader utils

AiReflect() return an **AtVector**, takes reference arguments:

```
AiReflect(&n, &i, &r) -> r = AiReflect(n, i)
```

AtShaderGlobals

Light sampling related shader globals have been removed: **Li**, **Liu**, **Lo**, **Ldist**, **Ld**, **Lp**, **i**, **n**, **we**. These are available as members of an **AtLightSample**, see Light Sampling section.

The **sx** and **sy** globals 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:

```
sx = -1 + (sg->x + sg->px) * (2.0f / AiNodeGetInt(AiUniverseGetOptions(), "xres"));
sy =  1 - (sg->y + sg->py) * (2.0f / AiNodeGetInt(AiUniverseGetOptions(), "yres"));
```

The **area** global was also removed, use **AiShaderGlobalsArea()** instead:

```
sg->area -> AiShaderGlobalArea(sg)
```

AtBBox

The **AtBBox** related functions are now class member functions:

```
AiBBoxInit(bbox, f) -> bbox.init(f)
AiBBoxIsEmpty(bbox) -> bbox.isEmpty()
```

AiBBoxUnion() and **AiBBoxIntersection()** now return the result **AtBBox**:

```
AiBBoxUnion(bbu, bb1, bb2) -> bbu = AiBBoxUnion(bb1, bb2)
AiBBoxIntersection(bbi, bb1, bb2) -> bbi = AiBBoxIntersection(bb1, bb2)
```

Tickets

These are the tickets related to porting to Arnold 5:

- ~~#5556: **`AtParamValue`** broken in Python bindings~~
- #5573: Restore **`AiRGBGamma()`** and **`AiRGBAGamma()`**

Opacity must now be premultiplied into the weight of BSDFs, BSSRDFs and other closures. For surface closures, the general rule is that the sum of all closure weights should not exceed 1 for the shader to be energy conserving.

```
closures.add(AiOrenNayarBSDF(sg, Kd, sg->Nf));
```