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.