OSL

OSL Arnold 5.0 OSL Arnold C++ API

OSL .osl Arnold 1

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);
}
```

`gamma` `gamma.osl` Arnold C++ `Cin` `gamma` `Cout`

Arnold .osl .oso .oso .osl

Arnold `oslc`

```
$ oslc gamma.osl
Compiled gamma.osl -> gamma.oso
```

.oso CPU OSL .osl

.osl .oso .so .dll Arnold

`kick` Arnold

```
$ 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
```

OSL `osl`

`osl` `shadername` .osl .oso `shadername` OSL `param_`

```
osl
{
 name myshader
 shadername somefolder/test
 param_value 0.5
}
```

.OSL .OSO

```
osl
{
 name myshader
 code "shader test(float value = 0, output float result = 0)
       {
          result = value * 10;
       }"
 param_value 0.5
}
```

OSL Arnold Arnold 1 1

OSL	Arnold
int	INT
int ()	BOOLEAN
int ()	ENUM
float	FLOAT
color	RGB
color	RGBA
point	VECTOR
vector	VECTOR
normal	VECTOR
point	POINT2
matrix	MATRIX
	ARRAY
closure color	CLOSURE
struct	POINTER

boolean enum OSL

```
#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)
        ...
}
```

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);
```

getattribute()

"geom:type"	string	
"geom:name"	string	
"geom:bounds	point[2]	(minmax)
"geom:objbounds"	point[2]	(minmax)

getattribute()

"camera:screen_window"	int[2]	
"camera:pixelaspect"	float	Pixel Aspect Ratio
"camera:projection"	string	
"camera:fov"	float	
"camera:clip_near"	float	
"camera:clip_far"	float	
"camera:clip"	float[2]	
"camera:shutter_open"	float	
"camera:shutter_close"	float	
"camera:shutter"	float[2]	
"camera:screen_window"	float[4]	(xminminxmaxymax)

Arnold PuvNNgtime OSL C++ API

Ps ()surfacearea()dtime dPdtime

Ci P

texture() gettextureinfo() texture() colorspace

Arnold C++ API OSL v 1 - v UDIM floor(v) + 1 - mod(v, 1)

```
// look up texture color
color tex = texture(filename, u, v);

// query texture resolution
int resolution;
gettextureinfo(filename, "resolution", resolution);
```

texture3d()

```
point Po = transform("object", P);
float density = texture3d("density", Po, "interp", "bicubic");
```

C++ API

```
closure color emission();
closure color background();
closure color diffuse(normal N);
closure color oren_nayar (normal N, float sigma);
closure color oren_nayar(normal N);
closure color translucent(normal N, float sigma);
closure color translucent(normal N);
closure color sheen(normal N, float roughness);
closure color metal(string distribution, normal N, vector U,
                    color n, color k,
                    float xalpha, float yalpha);
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 randomwalk_bssrdf(vector mfp, color albedo, float g);
closure color volume_absorption();
closure color volume_emission();
closure color volume_henyey_greenstein(color absorption, color scattering,
                                       color emission, float g);
closure color volume_matte();
```

:

```
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

OSL `trace()` `shade` `traceset` inclusive traceset traceset `-` exclusive traceset

`hithitdistPNuv` `getmessage()`

```
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

OSL *raytype(<ray type>)* 1 0

'camera"
"shadow"
'diffuse_transmit"
'specular_transmit"
'volume_scatter"
'diffuse_reflect"
'specular_reflect"
'subsurface"

OSL C++ OSL C++ (1 2 %)OSL OSL

OSL

OSL_OPTIONS

```
# 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"
```

- Open Shading Language
- Open Shading Language