# Open Shading Language

Larry Gritz Clifford Stein Chris Kulla Alejandro Conty
Sony Pictures Imageworks*

## 1 Introduction

Open Shading Language (OSL) was developed by Sony Pictures Imageworks for use in its in-house renderer used for feature film animation and visual effects. OSL's specification and full implementation have been released as open source software. [Sony Pictures Imageworks 2010]

OSL has syntax similar to C, as well as other shading languages. However, it is specifically designed for advanced rendering algorithms and has features such as radiance closures, BSDFs, and deferred ray tracing as first-class concepts.

OSL shaders can be organized into networks, with named outputs of nodes being connected to named inputs of other downstream nodes within the network. These connections may be done dynamically at render time, and do not affect compilation of individual shader nodes. Furthermore, the individual nodes are evaluated lazily, only when their outputs are "pulled" from the later nodes that depend on them (shader writers may remain blissfully unaware of these details, and write shaders as if everything is evaluated normally).

## 2 Radiance Closures

Traditionally, shaders compute just the surface color visible from a particular direction. These are "black boxes" that a renderer can do little with but execute to find this one value (for example, there is no way to discover which directions are important to sample). Furthermore, in other languages, the physical units of lights and surfaces are often underspecified, making it very difficult to ensure that shaders are behaving in a physically correct manner.

OSL's surface and volume shaders compute an explicit symbolic description, called a *closure*, of the way a surface or volume scatters or emits light, in units of radiance. These radiance closures may be evaluated in particular directions, sampled to find important directions, or saved for later evaluation and re-evaluation. This new approach is ideal for a physically-based renderer that supports ray tracing and global illumination. There are different types of closure primitives for BSDF, BSSRDF, emission, and volume scattering. The integrator knows how to handle them properly and each closure type may have different sets of methods internally. But shaders may combine them, manipulate them uniformly, and behave as if they all are simply returning an exitant radiance.

There are no "light loops" or explicitly traced rays in OSL shaders. Effects that would ordinarily require explicit ray tracing, such as reflection and refraction, are simply part of the radiance closure and look like any other BSDF. OSL does not have separate surface and light shaders; lights are simply surfaces that are emissive, and all lights are area lights. The radiance closures generalize both scattering and emission. You also don't need to explicitly set opacity variables in the shader. Transparency is just another way for light to interact with a surface, and is included in the main radiance closure computed by a surface shader.

The radiance closures produced by shaders are passed to a part of the renderer called an *integrator* that evaluates the closures (and lights) to determine which directions are important, and to compute the exitant radiance in the viewing direction. Advantages of this approach include that integration and sampling may be batched or re-ordered to increase ray coherence; a "ray budget" can be allocated to optimally sample the BSDF; the closures may be used for multi-importance sampling, bidirectional ray tracing, or Metropolis light transport; and the closures may be rapidly re-evaluated with new lighting without having to re-run the shaders.

## 3 Key runtime technologies

#### AOVs are specified using "light path expressions"

Production users often output many images containing partial lighting components such as specular, diffuse, reflection, individual lights, etc. In other languages, this is usually accomplished by adding a plethora of "arbitrary output variables" (AOVs) the shaders that collect these individual quantities.

OSL shaders need not be cluttered with any code or output variables to accomplish this. Instead, there is a regular-expression-based notation for describing which light paths should contribute to which outputs. For example, "CD+L" isolates just the indirect diffuse illumination, and "CS+D*L" isolates just reflections and refractions. If you desire a new output, there is no need to modify the shaders at all; you only need to tell the renderer the new light path expression.

#### No "uniform" and "varying" keywords in the language

In our OSL runtime implementation, shaders are evaluated in SIMD fashion on many points at once, but there is no need to burden shader writers with declaring which variables need to be uniform or varying. In OSL, this is done both automatically and dynamically, meaning that a variable can switch back and forth between uniform and varying, on an instruction-by-instruction basis, depending on what is assigned to it and the current conditional state.

#### Automatic differentiation for computed derivatives

OSL allows shaders to take derivatives of any computed quantity, and to use arbitrary computations as texture coordinates and expect correct filtering. In our runtime implementation of OSL, derivatives are computed using *automatic differentiation* utilizing dual arithmetic [Piponi 2004], computing partial differentials for the variables that lead to derivatives, without any intervention required by the shader writer. This has advantages over finite difference methods: it does not require that shaded points be arranged in a rectangular grid (or have any particular connectivity – very important for ray tracing), or require that any extra points be shaded. It also means that it is safe to use derivatives inside conditional statements and loops.

## References

PIPONI, D. 2004. Automatic differentiation, c++ templates, and photogrammetry. *journal of graphics, gpu, and game tools 9*, 4, 41–55.

SONY PICTURES IMAGEWORKS, 2010. http://code.google.com/p/openshadinglanguage. Web site.

*email: {lg,cstein,ckulla,aconty}@imageworks.com

Copyright is held by the author / owner(s).
SIGGRAPH 2010, Los Angeles, California, July 25 – 29, 2010.
ISBN 978-1-4503-0210-4/10/0007