

# OSL Shaders



The OSL shaders used in the above examples can be found [here](#).

It is possible to create your own shaders using [Open Shading Language](#). OSL shaders can be used to implement anything from simple texture patterns to full materials using closures. They can be placed in the plugin search path, and will then be automatically loaded like other types of shaders. Once loaded, they can be inspected, instantiated, and linked in the same way as C++ shaders. The `osl` node provides an alternate way to load OSL shaders, which can be used to write shader code for a specific material. When setting the shader name or code, the parameters from the OSL shader automatically appear on the node to be set or linked.

 More information about using OSL shaders can be found [here](#).



A video that shows how to use the OSL shader can be found [here](#).

An example `.osl` shader and `.mtd` file can be found [here](#).

## Limitations

- The shaders must have unique names. If the name conflicts with an existing shader, the OSL shader won't load. You should see a warning in the log in this case.
- The name of the output attribute is not the same as the shader code.

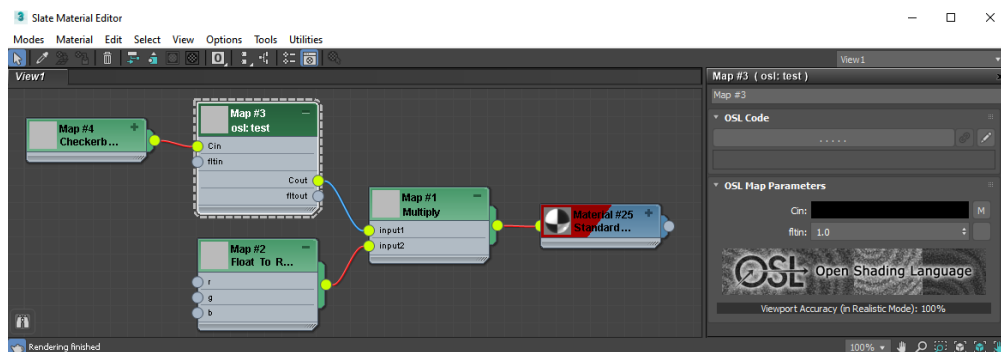
- Unable to do #includes in OSL shader code.
- No support for exposing input attributes of type Array and Matrix.

## Multiple Outputs

Shader outputs can be linked to inputs of other shader nodes in an `.ass` file or programmatically via the `AiNodeLinkOutput()` function by optionally specifying which output is desired when linking. When no output is specified, the node's default output parameter will be chosen, preserving Arnold's existing behavior. Standard OSL language rules apply when specifying multiple OSL shader outputs. An example OSL shader with three color outputs could look like the following snippet and each output could be connected separately :

```
shader test_shader
(
    output color result = color(1, 1, 1),
    output color good = color(0, 0, 1),
    output color bad = color(1, 0, 0)
)
{
}
```

An example `.ass` file using multiple outputs can be downloaded [here](#).



OSL shader using multiple shader outputs

## Installation

Just like any other third-party shader libraries, OSL shaders placed in the shader search path are automatically registered as Arnold shader nodes. The OSL shader parameters are converted to Arnold parameters. Once loaded, they can be inspected, instantiated and linked in the same way as C++ shaders.

To render OSL shaders directly with Arnold, you must do the following:

- Put the OSL shader (`.osl` and `.mtd`) in a folder, and set `ARNOLD_PLUGIN_PATH` to point to this folder. For example:

ARNOLD\_PLUGIN\_PATH=C:\shaders\osl

- Arnold will automatically compile the .osl file and produce a dll (.oso) file.

### OSL in 3DS Max 2019

In 3ds 2019, there is a new way of using OSL. Arnold will take advantage of this automatically. Simply use the OSL maps provided with 3ds max, or use the OSL editor built into the OSL map to write your own, or load up external OSL files. In this mode, the actual OSL sources are embedded into the Arnold scene and exported as text into an .ass file, so there will be no dependency of external OSL files when rendering, and the limitations about shaders not having the same name are lifted. You could, in theory, have twenty totally different shaders named 'Bob' and it would still work (not that we recommend this).

See the 3ds Max documentation for more information about OSL. There is also a GitHub for OSL shader exchange which also contains developer documentation and more info at: <https://github.com/ADN-DevTech/3dsMax-OSL-Shaders> ”