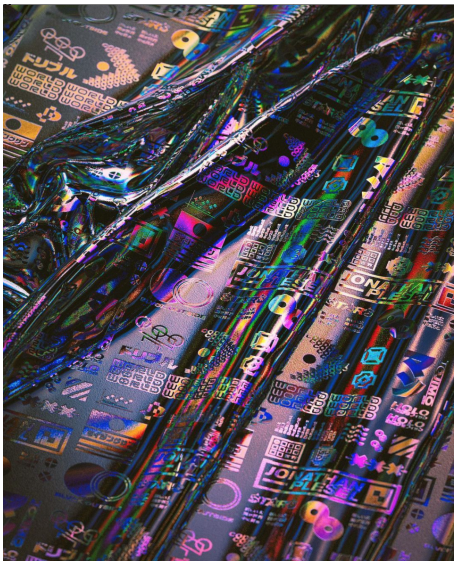
Final Project Report: Implementing Phong Shading for Blender's Cycles Renderer

Motivation and Impact

While experimenting with different textures and techniques in Blender, visual artist Intton Godelg came across a limitation. He discovered that there is no straightforward way to create holographic materials with shaders while using Blender's Cycles renderer. Upon browsing forums and Blender designer communities such as NPR Shaders and Trippy Visuals, he came across several references to iridescent materials and how to create those in Cycles. However, he was unable to find a simple way to create holographic materials instead.

To the untrained eye, these two types of texture might look the same, however, they are not. An iridescent material changes color based on the angle of view and the angle at which the light hits the surface. Meanwhile a holographic material changes color based on the amount of light hitting each point in the surface, regardless of the angle of the viewer. Being able to create holographic materials that can be rendered in Cycles opens a plethora of creative possibilities for artists who want to explore visuals with realistic looking holographic materials such as security seals and electronics.



Holographic Texture by @jonathanplesel

Iridescent Texture by @intton

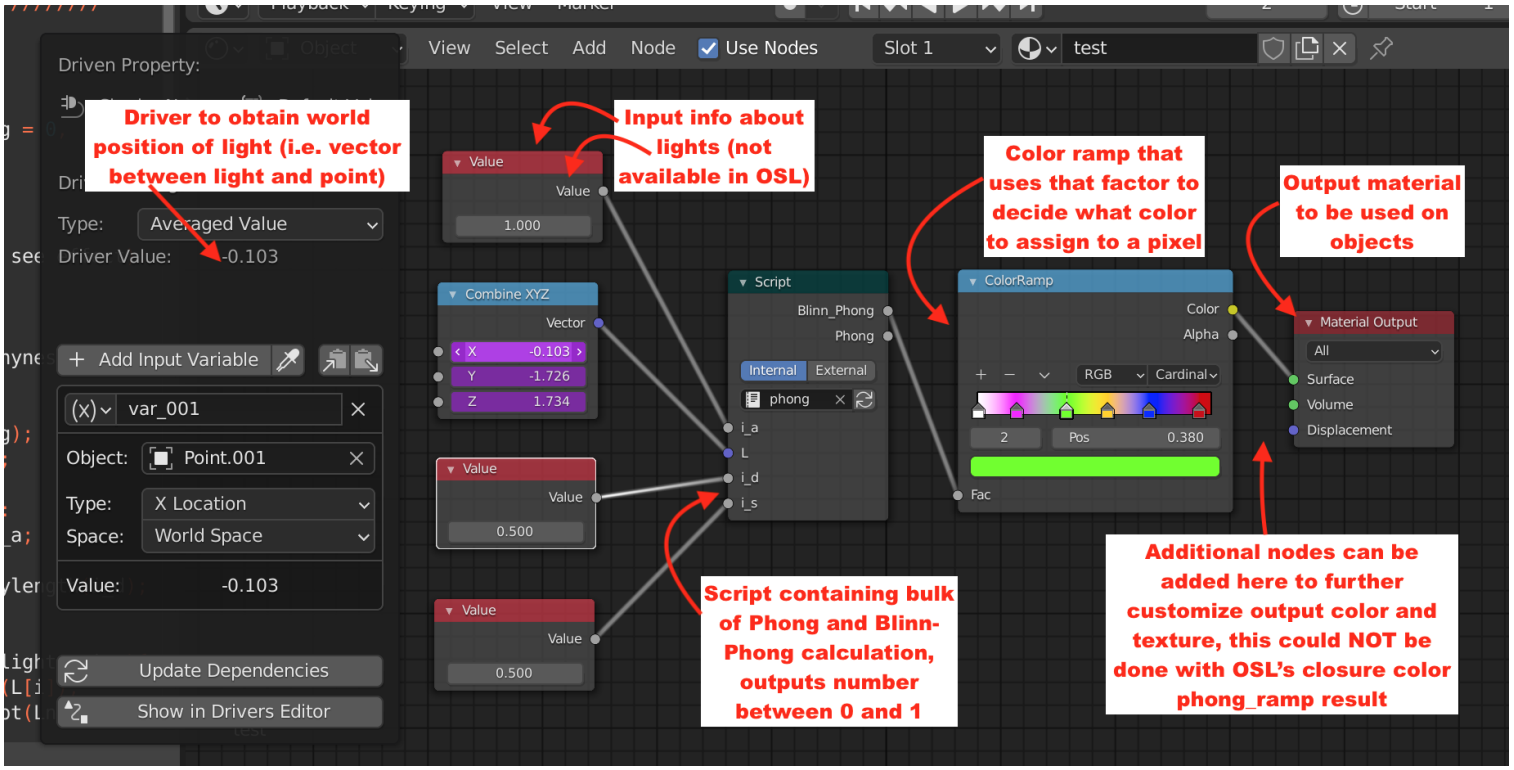
It is possible to create holographic materials using Blender's other renderer Eevee, as well as Unity's Octane. There are also a few workarounds used by several Blender designers that are able to approximate holographic materials in Cycles. Nevertheless, the only approach that seems to actually generate a material that changes color based on the amount of light hitting its surface is the Open Shading Language shader, `phong_ramp`.

Unfortunately, `phong_ramp` has a couple of drawbacks. First, it has very poor documentation. It seems to be a black-box with just a few brief mentions across the web showing examples of use, but no implementation details. Second, it lacks flexibility because it outputs a `closure color` type, which is a function to be called by the renderer when light data is available (as OSL is completely ignorant of light). This means that the output color cannot be manipulated, and its components cannot be accessed, which limits the level of control a designer can have.

Approach and Implementation Details

Instead of attempting to modify the mysterious `phong_ramp` shader to output a more malleable data type such as a `float` factor, my approach was to create a Phong ramp from scratch by implementing Phong's shading formula directly through a Script node (programmed in OSL) and other auxiliary nodes in Blender.

In overview, we can use Phong's formula to output a `float` number between 0 and 1, this number can then be used as a factor in a Color Ramp node, that can in turn be used as a material's surface color. This connection of nodes, including the code for the Script node, can be reviewed in the attached file *phong.blend* -- refer to the README for further details.



Phong's formula for the script node shown above is as follows:

$$I_{\mathrm{p}} = k_{\mathrm{a}} i_{\mathrm{a}} + \sum_{m \in \text{ lights}} \left(k_{\mathrm{d}} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_{\mathrm{s}} (\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,\mathrm{s}} \right)$$

, where $k_a i_a$ is the ambient light component, $k_d(L \text{ dot } N) i_d$ is the diffuse light component, and $k_s(R \text{ dot } V)^{\alpha} i_s$ is the specular light component. The script code in *phong.blend* has been annotated to explain what each of the individual variables stands for.

To make this formula more computationally efficient, we can avoid calculating the reflection vector R by replacing $(R \text{ dot } V)$ with the Blinn approximation $(N \text{ dot } H)$ where:
H = the halfway vector between L and V, i.e. $L + V / 2$

Now, N and V are built in variables in OSL; R can be calculated from L (or omitted using Blinn-Phong); and k_a, k_d, k_s, and α are all constants that can be hard-coded by the user.

This leaves us with L, i_a, i_d, and i_s which need to be obtained from additional Blender nodes. The challenge here is that OSL is unaware of lights and lighting in the scene, so we need a way to tell our script how many lights there are in the scene, and what their positions and intensities are.

How to get L: For the scenario where there is only a single point light in the scene, L can be easily obtained by getting the world coordinates of the light source using Blender Drivers on a Combine XYZ node. However, things become complicated when there exist multiple light sources, or when the lighting comes from ambient sources, such as an HDRI, rather than specific points. My final approach focuses only on the simplified single point light scenario.

How to get i's: I was unable to find a way to obtain this information using Blender's existing nodes, at least with the amount of time remaining before this project is due. However, I included the value nodes with some reasonable approximations in the hopes of finding a way to obtain these values in the future.

Results

After implementing the Blender nodes and Phong shading script described in the previous section, I was able to set the material of an object (Suzanne) in Blender that changes color based on the amount of light each point gets. My script node is able to calculate both Phong and Blinn-Phong shading as a factor from 0 to 1 that can then be used as the input of a Color Ramp node. Blinn-Phong seems to give a nicer looking result, and is computationally more efficient.

Here's a link to a video showing the render view of my Suzzane:
https://youtu.be/hIcesa82Hnk - I recommend watching it at 2x speed.

Video timestamps:
0:00 - 0:21 - Demo on how the color of the material changes when the position of the light (and hence the amount of light at each point) changes.

0:21 - 0:57 - Shows what happens when varying the light intensity variables -- these variables should ideally come from actual light sources in the scene, instead of user input, but this is a current Blender limitation.
When the ambient intensity is higher, the general colors of the material tend towards the 0 side of the color ramp. Meanwhile, when the diffuse intensity is higher, the diffuse (less directly exposed) areas of the material become more pronounced, and when the specular intensity is higher, the specular (more directly exposed) areas of the material become more pronounced.

0:57 - 2:36 - Changes shading from Blinn-Phong to Phong, and quickly shows how colors look different between the two when changing variables around. In particular, Phong seems to have a more uniform distribution between all colors, while Blinn-Phong seems to be heavier on the extremes.

2:36 - 3:43 - Explores changing the constant, hard-coded parameters: α for shininess, and k_a, k_d, and k_s for reflection constants. Namely, increasing α results on a sharper specular region

(i.e. a more "mirror-like" surface) while decreasing it results on a more diffuse or rough material. The other constants just accentuate their respective areas of lighting in the texture.

Challenge and Innovation

I believe this project was innovative because it addressed a limitation in Blender Cycles that has not been successfully addressed in the past. Namely, there is no recorded straightforward way to create holographic textures in Cycles, therefore any attempt to do so can be considered innovative.

On working on this project I came to realize that the reason why no one has come up with a straightforward way to create these types of materials, is because there actually *is* no straightforward way to do this with the current technical capabilities of Cycles and OSL.

Some of the challenges I encountered were:

1. Getting it to work for multiple lights / ambient lighting:

For the case when there were multiple point lights, I was able to write the OSL code for Phong's formula using arrays and loops (see commented out code in *phong.blend*'s script). However, I was unable to find a way to input the values from an arbitrary number of light sources as an array into the script node. The output factor seems to be correct when hard coding dummy *L* vectors and intensities, but there seems to be no functionality to get an actual array of vectors for all lights from the scene.

2. Getting light information in general:

OSL is built to generate `closure color`s which are data types that can be scaled and added to other `closure color`s, but cannot be manipulated or accessed componentwise. This is because they are not actual colors, instead they are a function that holds a set of conditions and will return an actual color given additional lighting and ray tracing information at rendering time. This is a limitation because it means OSL shaders are ignorant of light pre-render, so generating any value that needs light information but is not a closure becomes challenging if not impossible.
In addition, I was unable to find any node or value in Blender that represented the lights' intensity values (ambient, diffuse, and specular), so that part of the formula needed to be approximated.

3. Actually contributing to the open source project by adding this as a native shader:

My shader did not get close enough to be worthy of an open source contribution at the moment, but I will continue to work on it and hopefully will get something better in the future.

Despite the unresolved challenges I faced, I still believe this was a valuable experience to get more familiarized with OSL (a language I had never used before), as well as understanding a new computer graphics concept and formula: Phong shading. Initially it was a little difficult to navigate OSL's documentation and understand what `color closure`s are and why OSL works the way it does, but once I did I understood why the project I took on was not as easy as I expected it to be.

I believe I deserve **15 points** on this section, because even though I didn't get the results I was aiming for with my project proposal, I did get some very good results considering my beginner level and the technical limitations of Cycles and OSL. Nevertheless, I do recognize that if I had invested more than the 15 hours did in this assignment, I might have figured out how to tackle some of the challenges I faced.

Sources

Asking around about phong_ramp and the holographic materials:

- https://blender.community/c/rightclickselect/Cmfbbc/
- https://blenderartists.org/t/using-osl-phong-ramp-as-a-factor/1207588/2
- https://blender.stackexchange.com/questions/166626/using-phong-ramp-as-a-factor

Only documentation available about OSL's Phong Ramp:

- https://blog.michelanders.nl/2012/12/a-phong-ramp-car-paint-osl-shader-for_73.html

Phong and Blinn-Phong information:

- https://en.wikipedia.org/wiki/Phong_reflection_model -- formula used
- https://www.robots.ox.ac.uk/~att/index.html
- https://learnopengl.com/Advanced-Lighting/Advanced-Lighting

OSL language specs:

- https://weber.itn.liu.se/~stegu/TNM084-2019/osl-languagespec.pdf

People who have done similar things:

- https://bis.interplanety.org/library/materials/item/539

Getting light source vector:

- https://polycount.com/discussion/213338/solved-blender-2-8-shader-editor-is-there-a-light-direction-input-node