PxrSurface Mattes and Position

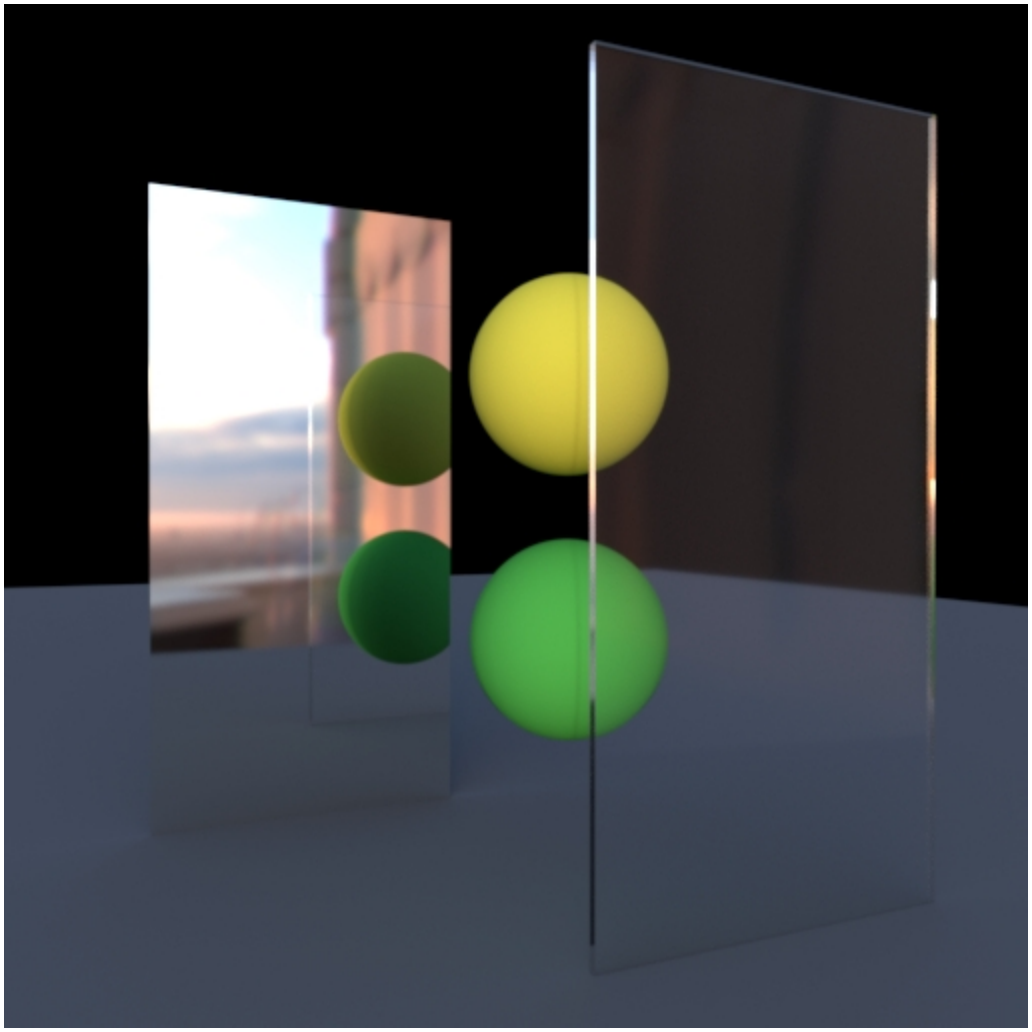
RenderMan has the option of using additional User Lobes for data during rendering. Shader writers can use these to pass information around and some useful lobes are included for users needing some common data.

> ⚠ This document relies on some familiarity with the tokens used for LPE and their effects on the light path collected.

PxrSurface has two added data user lobes. One is used to output World Position (U3) and another can be used to specify a color or pattern for output (U4). Neither of these lobes are rendered to the beauty and rely on the user to specify a Display Channel (AOV) for output. The bulk of this document will cover the use of the U4 lobe for color or patterns, but the LPE given can substitute U3 to get the World Position of the object.

The typical usage scenario is output for post operation and compositing. This data can be extracted from indirect effects. This makes it possible to write an AOV with a matte for an object in a reflection or transmission (refraction). This adds flexibility without having to render in layers. The equivalent would be RGB Mattes or a "Clown Pass", so-called because of the primary colors used.

Imagine a car headlamp rendered with the lights off. Using these mattes, one could extract the matte from the light from underneath the headlamp cover /lens and use a compositing package to "turn on" the light by using the mask to control glow and color and even animate this without re-rendering. Below we use a very simple example to show the results and some example Light Path Expressions to help you create your own. An example RIB file is included at the bottom of the page, please map your own HDRI to the Dome Light, the one used in the examples is the Griffith Observatory shipped in the Preset Browser assets. The scene has a pane of glass (with thickness) and a mirror.



Typically when extracting masks/mattes, you see the mattes for anything visible directly to the camera like the Cryptomatte specification. In this case I have matte colors assigned to the green and yellow balls and the pane of glass in the scene. I have chosen the typical RGB colors and assigned them to the User Color in the global parameters of PxrSurface. You can also drive these with object attributes for a procedural workflow.

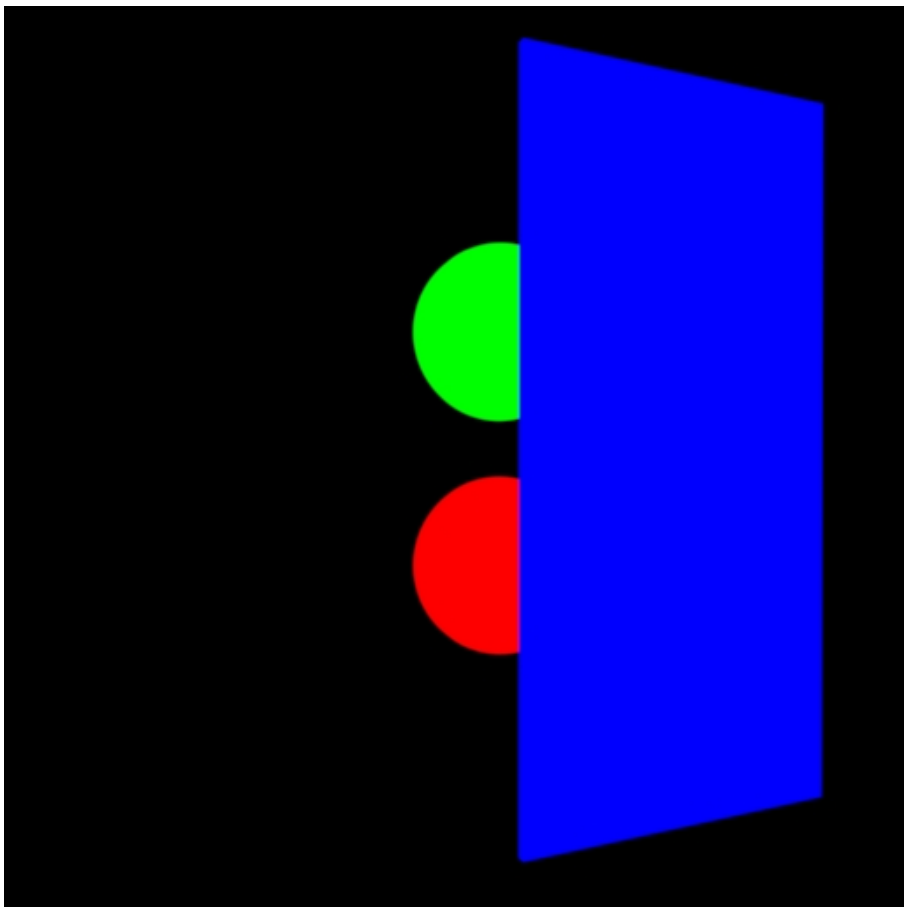
For data passes we make use of LPE modifiers, `nothruput;noinfinitecheck;noclamp;unoccluded;` you can find these at the bottom of the LPE page, for the purposes of this document you should pay most attention to the part of the LPE that begins with the camera (C) as it is used to define the path collected. The rest is included for completeness.

- `unoccluded` – returns unoccluded or unshadowed result. If not included, your result will include shadowing and may not be useful as a matte as the shape of the object and shadowing will be evident
- `noclamp` – returns unclamped result.
- `nothruput` – does not apply thruput (thruput is the accumulative albedo of the objects hit by rays). If omitted, the masks will be attenuated. For example, a color of 1,0,0 is pure red but may be 0.5,0,0 after a reflection or refraction, the actual result is based on the light path and material response(s)
- `overwrite` – instead of outputting the accumulated result, overwrite it. One example of using this is for the albedo output where we do not want an accumulated result. If left out your result may contain values different than expected. In this example the color 1,0,0 may return 6,0,0 etc. and bleed onto neighboring pixels
- `noinfinitecheck` – do not do any infinite check.

User Color

Let's start with the preset or usual defaults. You would get this result by default and is most common, it is also the result using the bridge product's default LPE preset for User Color in something like RenderMan for Maya:

```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;CU4L
```



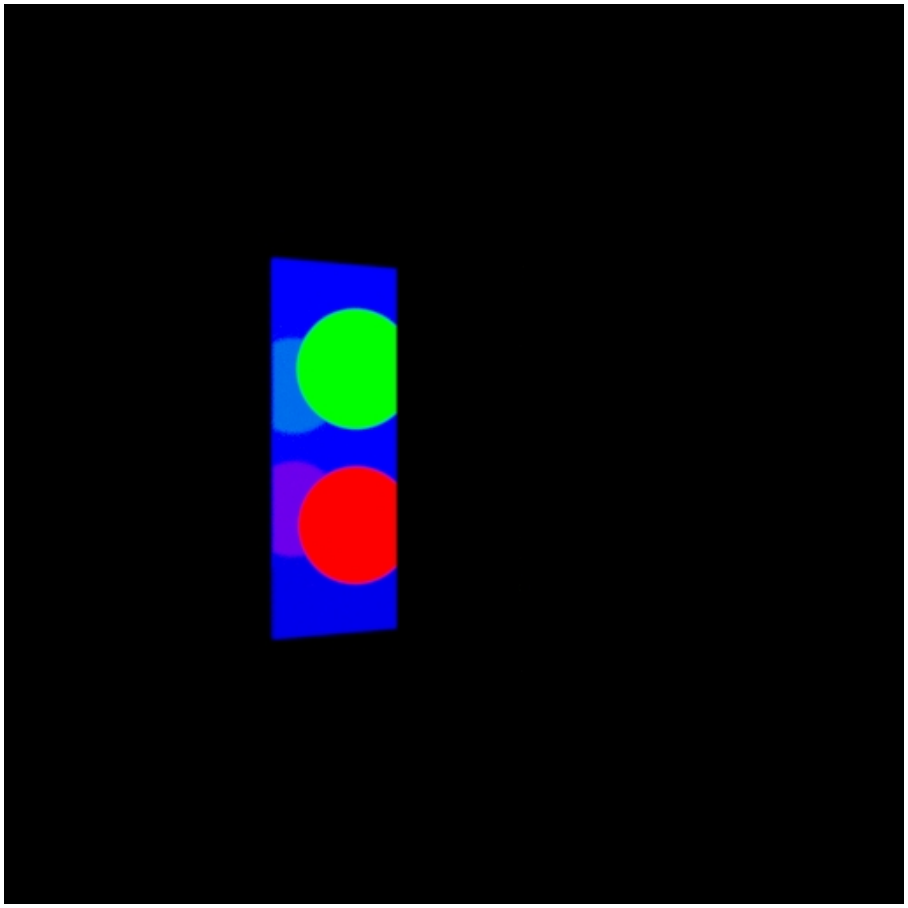
The lobe rendered is the U4 user lobe. The LPE is: camera to U4 lobe to light, no other paths.

In compositing this would be an issue if you needed to use the matte to adjust the balls in the scene because the matte through the glass is not visible to the renderer. Their reflection would also be unchanged without some rotoscoping work which can be tedious. And if the mirror or glass are warped, the manual option becomes quite difficult.

Using LPE and the User Color attribute you can begin to isolate the mattes. Let's start with the reflected mattes and see if we can isolate them using the following expression:

```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;C<RS>+U4[LO]
```

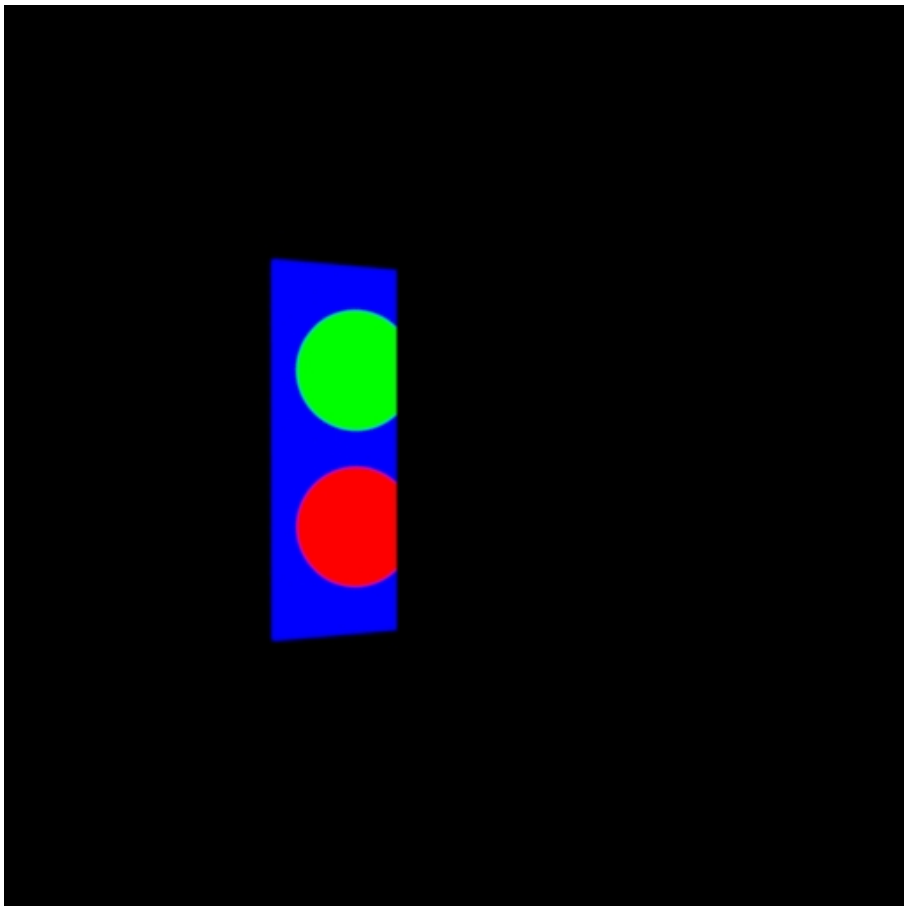
The above LPE (if you're familiar enough with the tokens on the main documentation page) collects only the reflected specular contribution of the User Color lobe after a bounce of like (the + denotes indirect) and you get the following result:



Using regular LPE syntax this result is basically correct, even the double reflections, but in the beauty image this secondary reflection isn't visible because of the HDRI, the balls just aren't bright enough. (However, the equivalent LPE of the mirror reflection will indeed show this secondary ball reflection not visible in the *beauty*.) This is not intuitive at first because you'd expect a similar response to what is seen in the beauty. But User Lobes are not energy conserving material responses, instead they are data. So we can remove this extra reflection easily by changing the LPE slightly:

```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;C<RS>U4[LO]
```

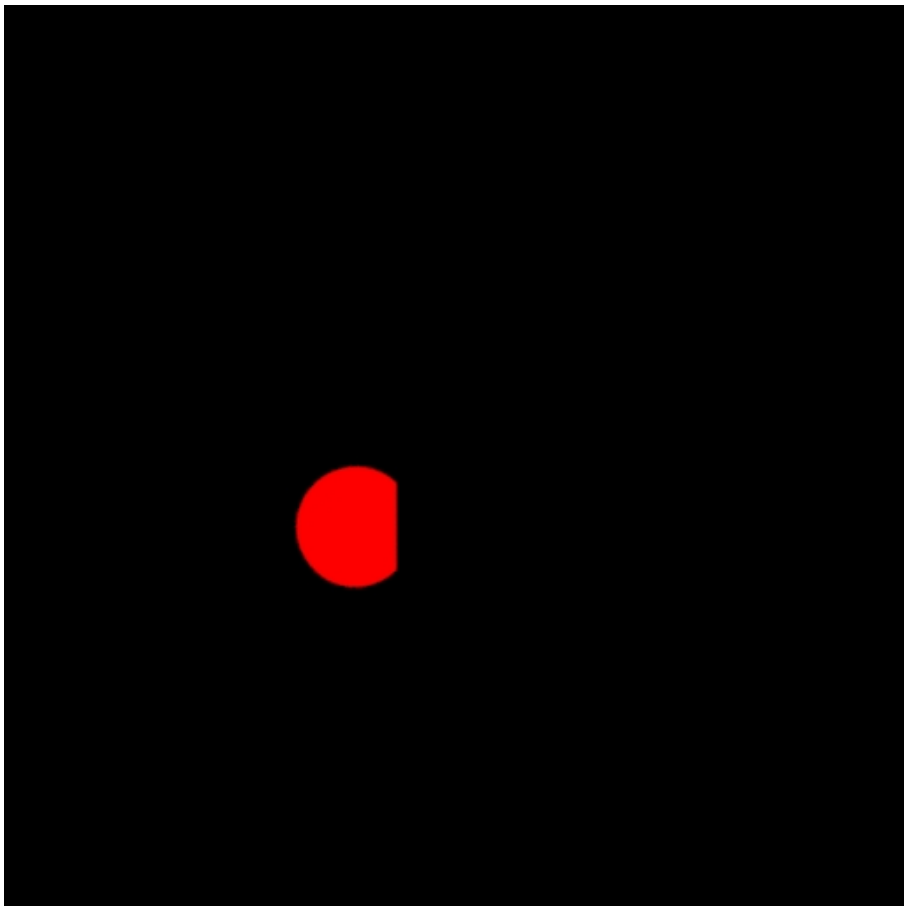
Removing the + used for indirect bounces you've asked instead to be given the first and only the first reflection of the user lobe. The result is a bit more what you might expect given the beauty render:



So now you can retrieve the user color from the reflection for all three materials using the a user color. However, you may need to isolate just a single object for more...extreme art direction. Using an LPE Group on the objects can help you achieve this with the standard LPE syntax. So let's grab the lower green ball which has been tagged with an LPE group. You'll see this LPE is quite similar:

```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;C<RS><.U4'greenBall'>[LO]
```

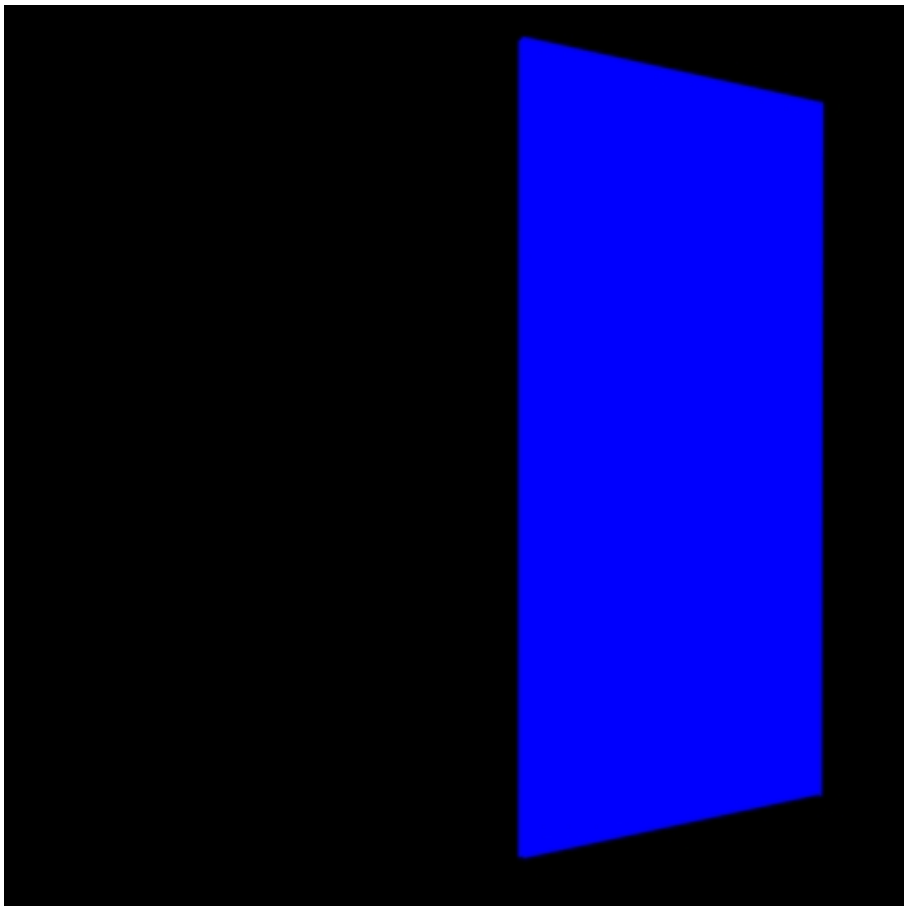
The result is:



So from above to now, we've begun to isolate not just the effect we need (the reflections) but the single object itself. From here you can begin to see how you can now create AOVs flexibly to handle your compositing needs.

Now, about that pane of glass, it presents a different set of issues we'll look at. Some of these LPE can be quickly adjusted to handle transmission (refraction) but note that the glass itself has a user color. If we render the transmission of the user lobes using the below LPE, you get a result that follows:

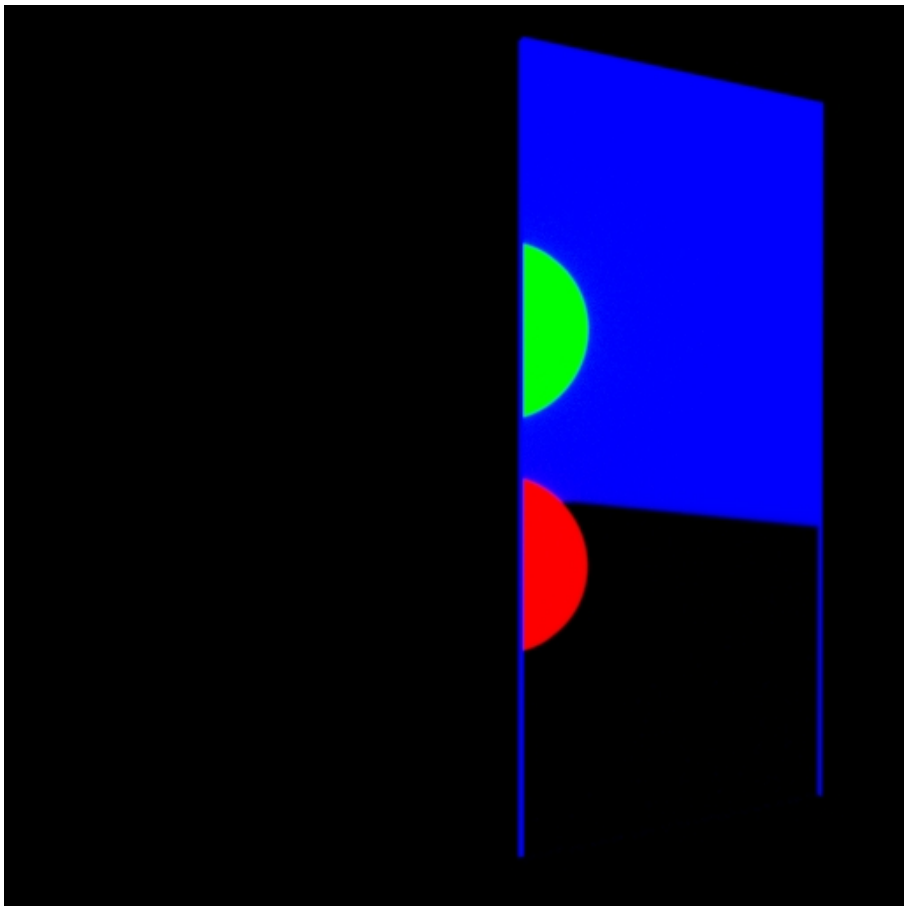
```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;C<TS>U4[LO]
```



Well, quite obviously you get the glass pane's user color as it refracts itself. But we can extend this to capture the indirect response, meaning after we've struck the glass with a primary ray. This means we use the + modifier for indirect again.

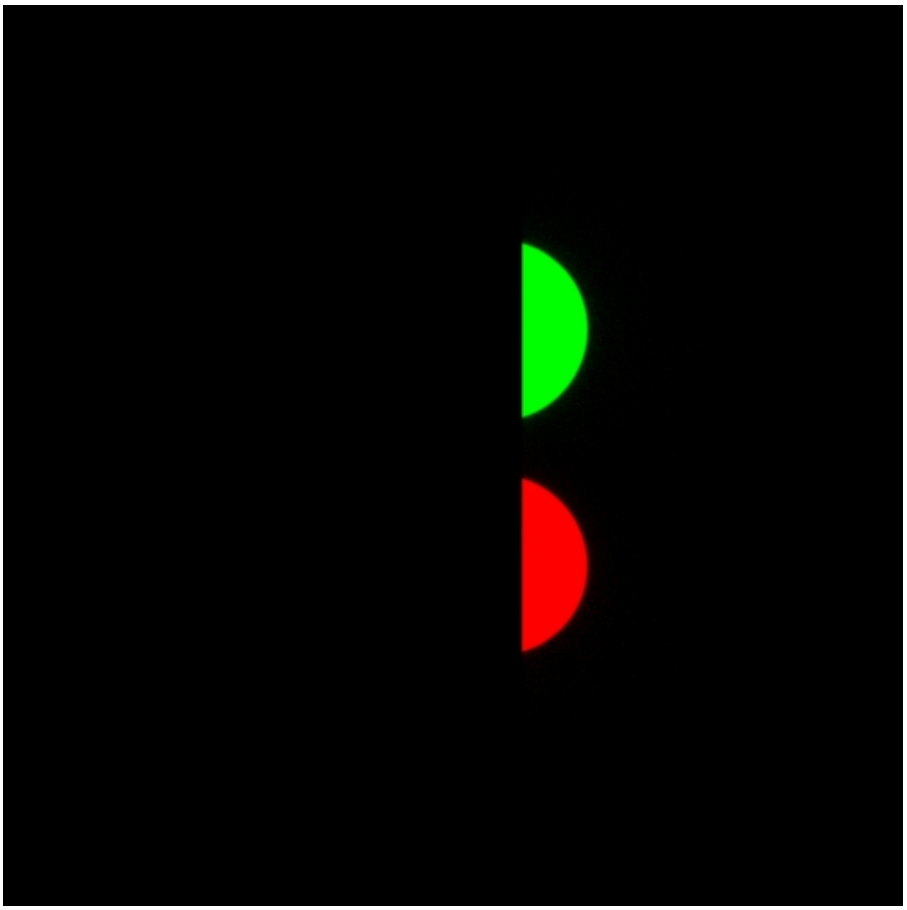
```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;C<TS>+<.U4>[LO]
```

The end result misses the front of the pane of glass (since we want indirect +) and gives you the transmitted balls and the backside of the glass against the environment. The ground plane has *no* user color so is rendered black.



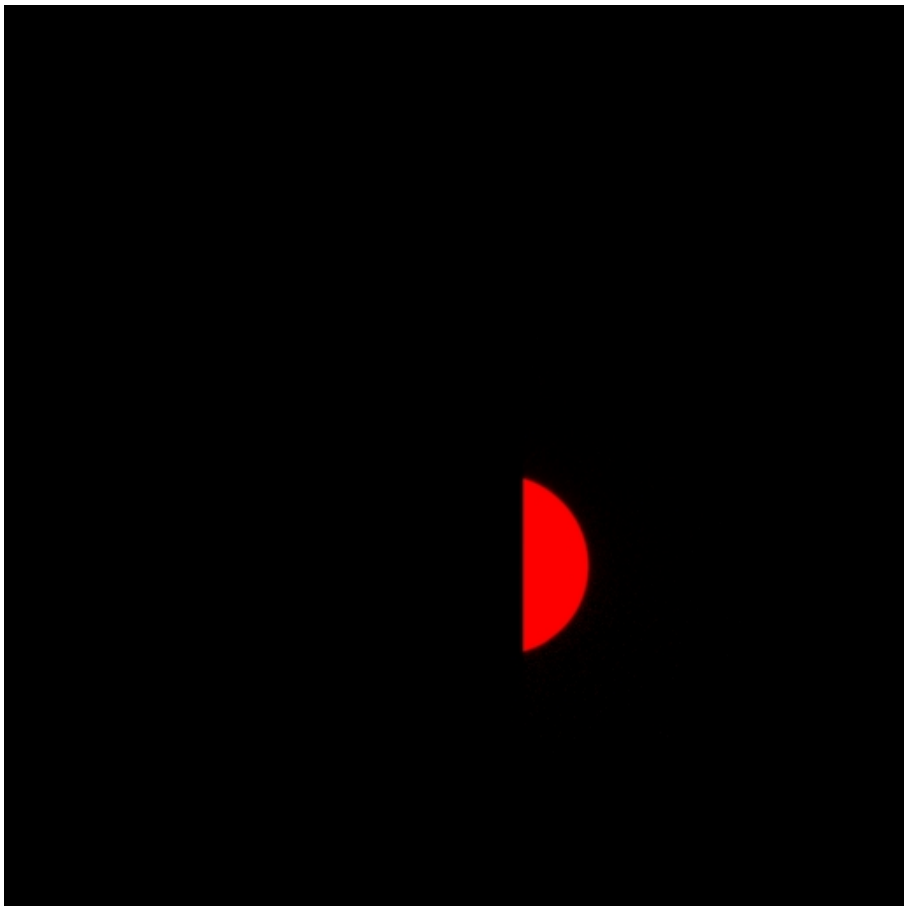
This could work, but we really want just the balls. So you can now use the exclude token, the caret ^, to remove the glass pane that was tagged in an LPE group.

```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;C<TS>+<.U4[^'glassPane']>[LO]
```



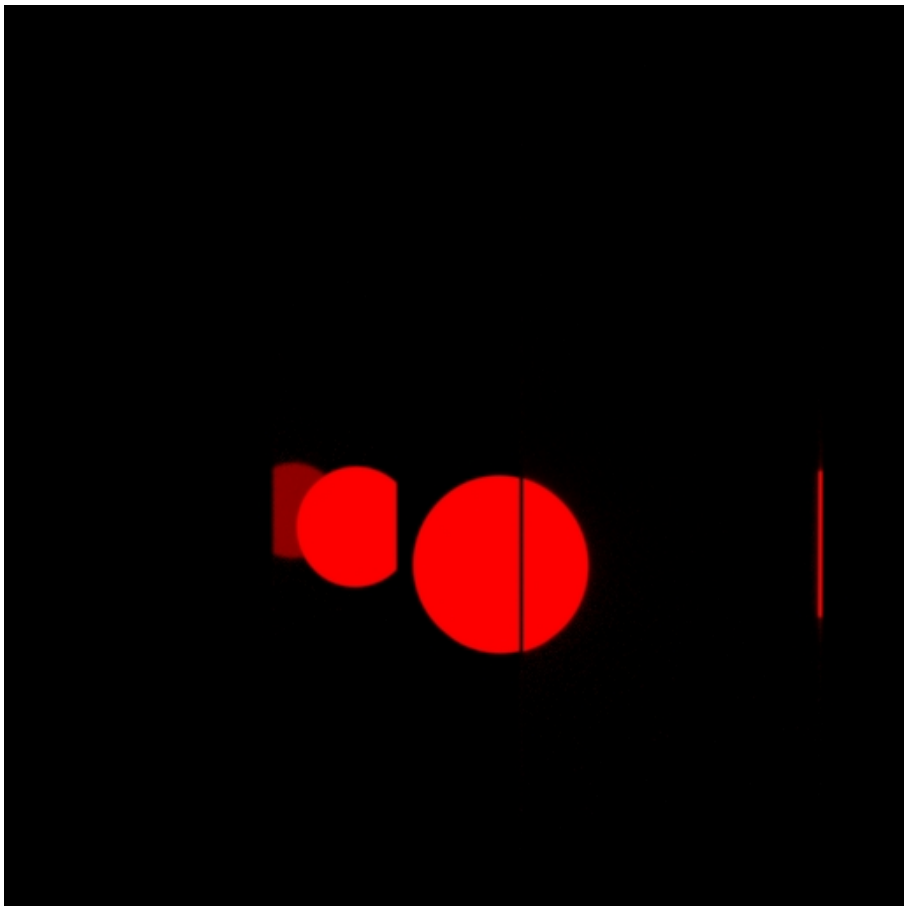
Again, you can refine this to just the lower green ball by saying you only want the greenBall

```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;C<TS>+<.U4'greenBall'>[LO]
```

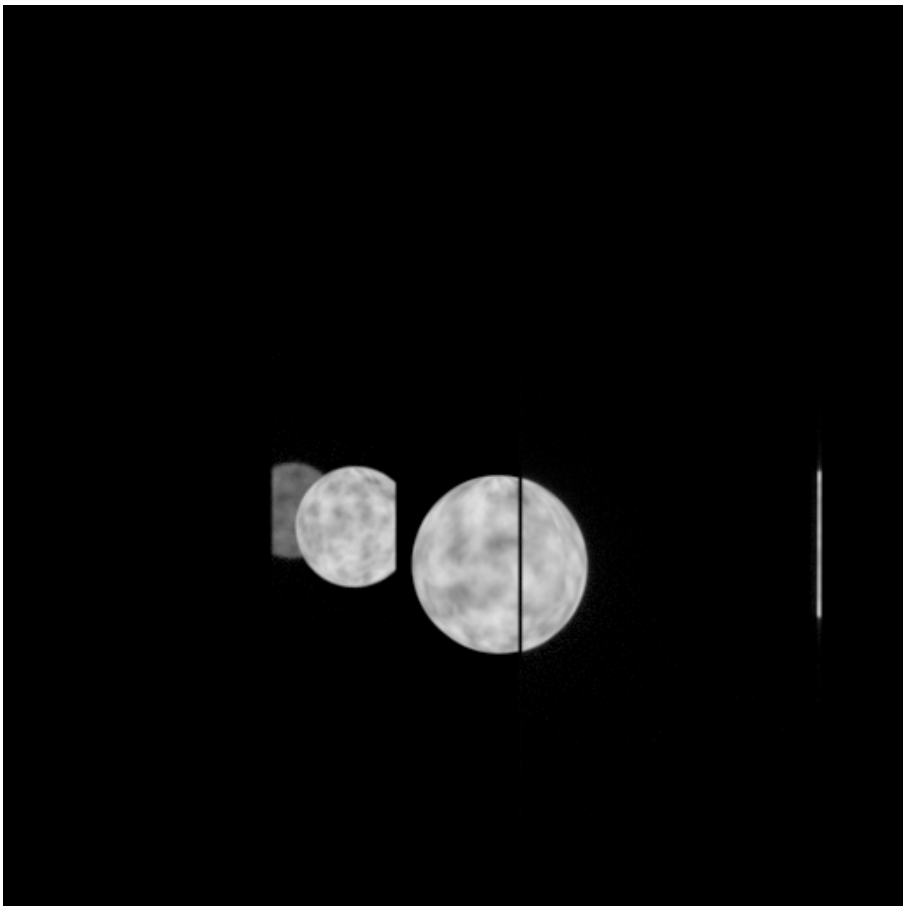


You can also somewhat indiscriminately grab all the specular <.S> effects of the matte from the green ball using the following LPE where * means all the bounces of light based on your render settings:

```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;C<.S>*<.U4'greenBall'>[LO]
```



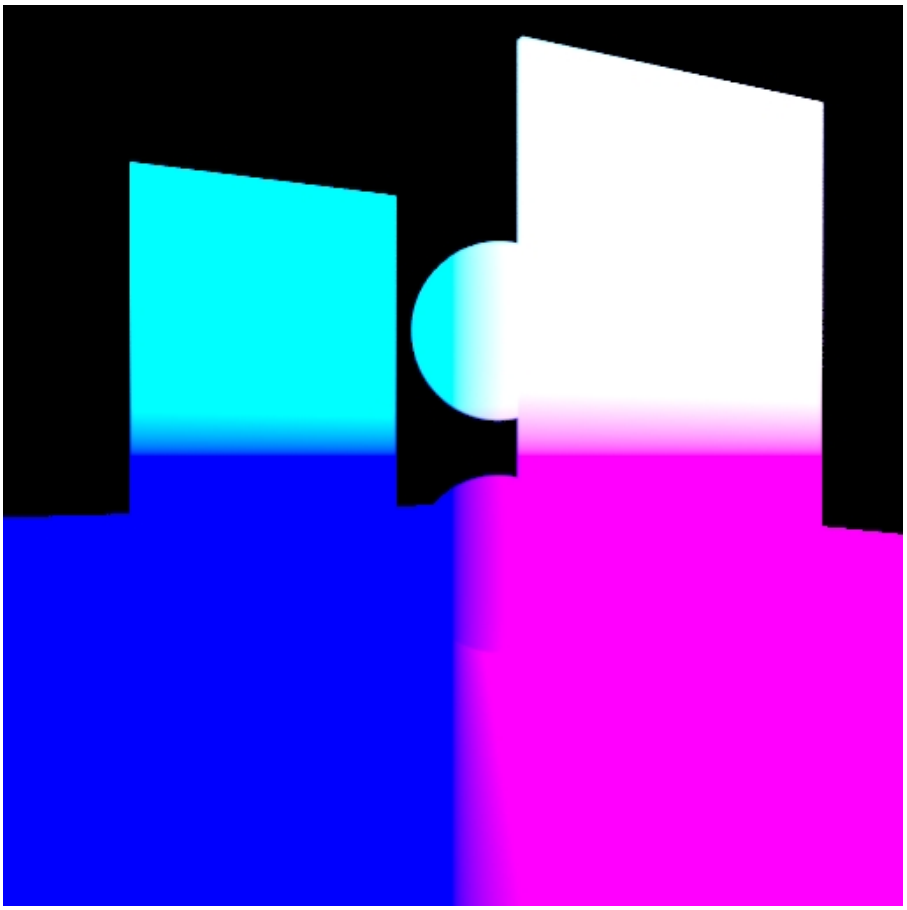
Using the same LPE you can also connect a pattern to the User Color like PxrVoronoise or even a texture:



World Position

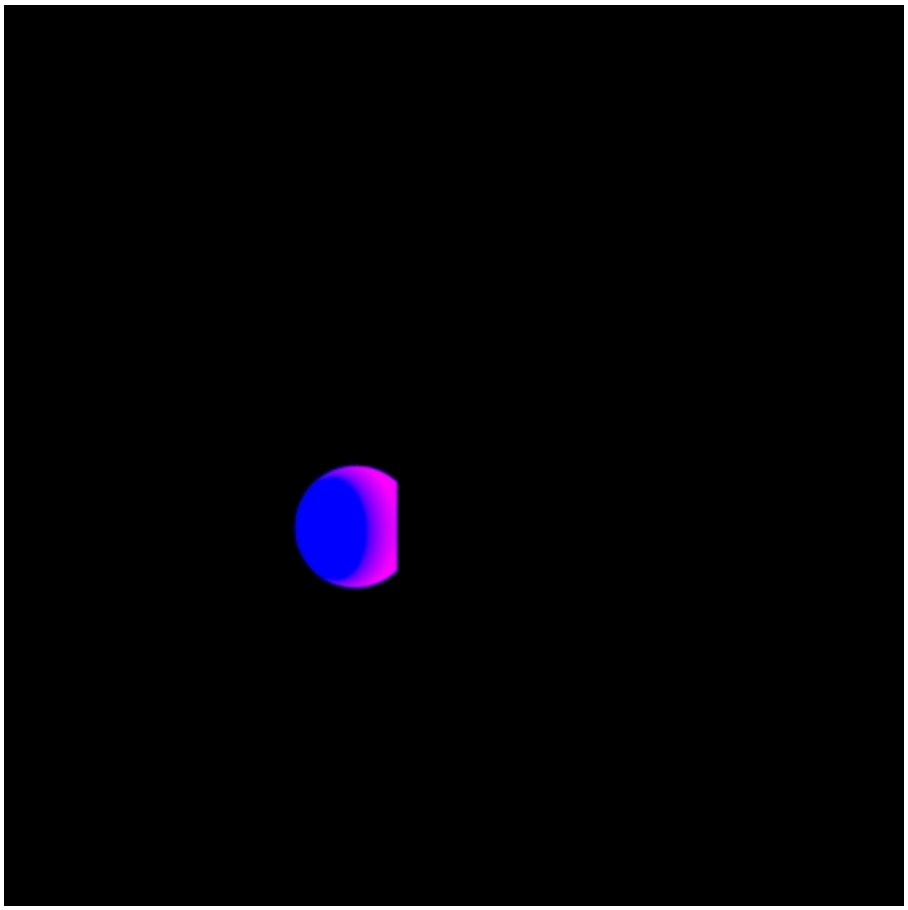
The default LPE for the Position pass results in this render, note that Position is enabled for anything with PxrSurface assigned.

```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;CU3L
```



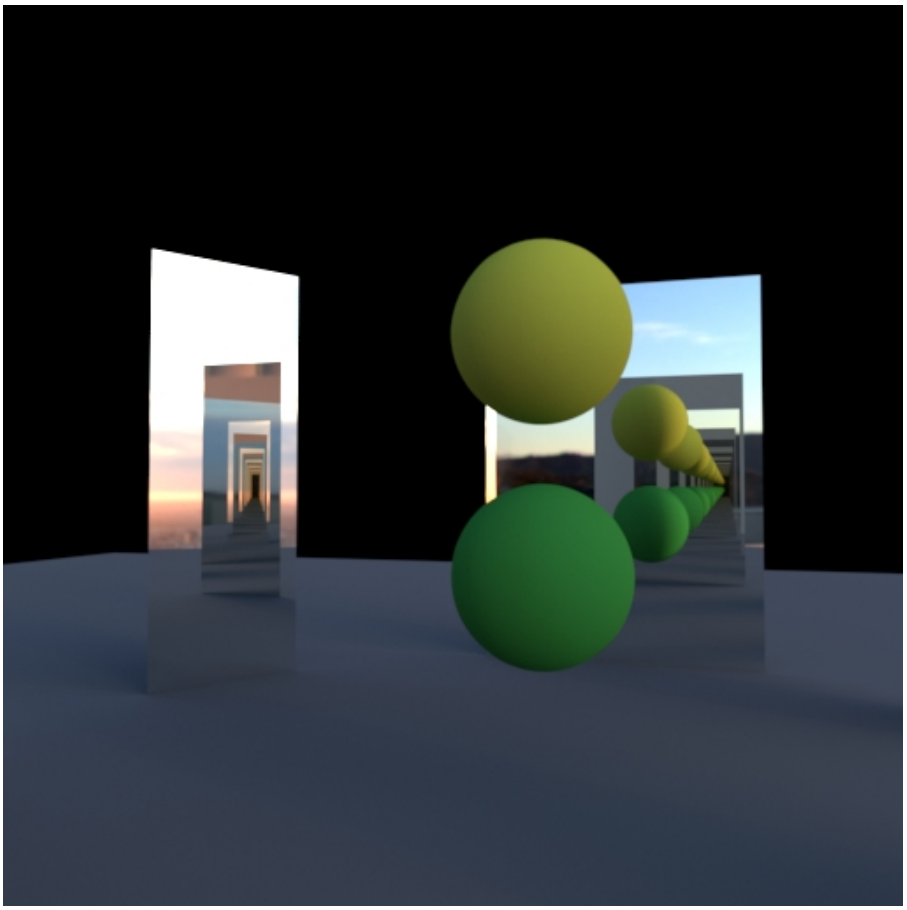
Using the same LPE as above you can begin to refine these results as well, for example the green ball in the mirror.

```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;C<RS><.U3'greenBall'>[LO]
```

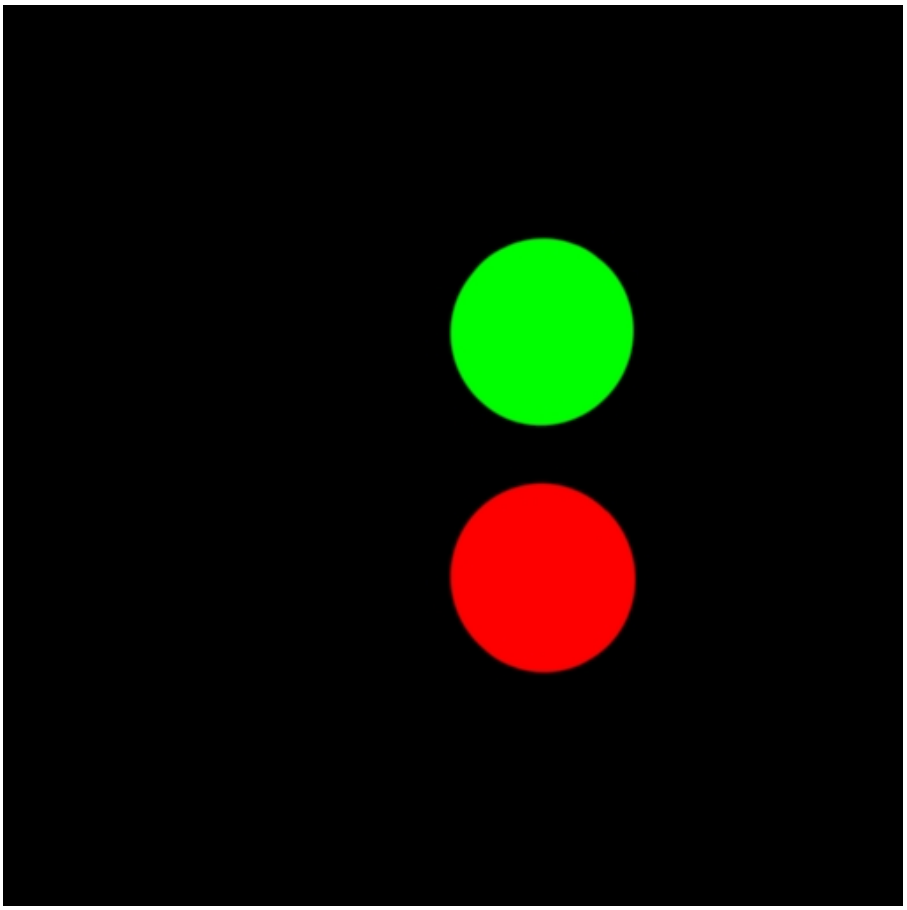


Advanced Thoughts

Since you can use the functions of LPE, you can do some advanced things like selecting the specific bounce of light for an object as well as the object where it's visible (reflected or transmitted). In this example, only the balls are using an assigned User Color to make this more obvious.

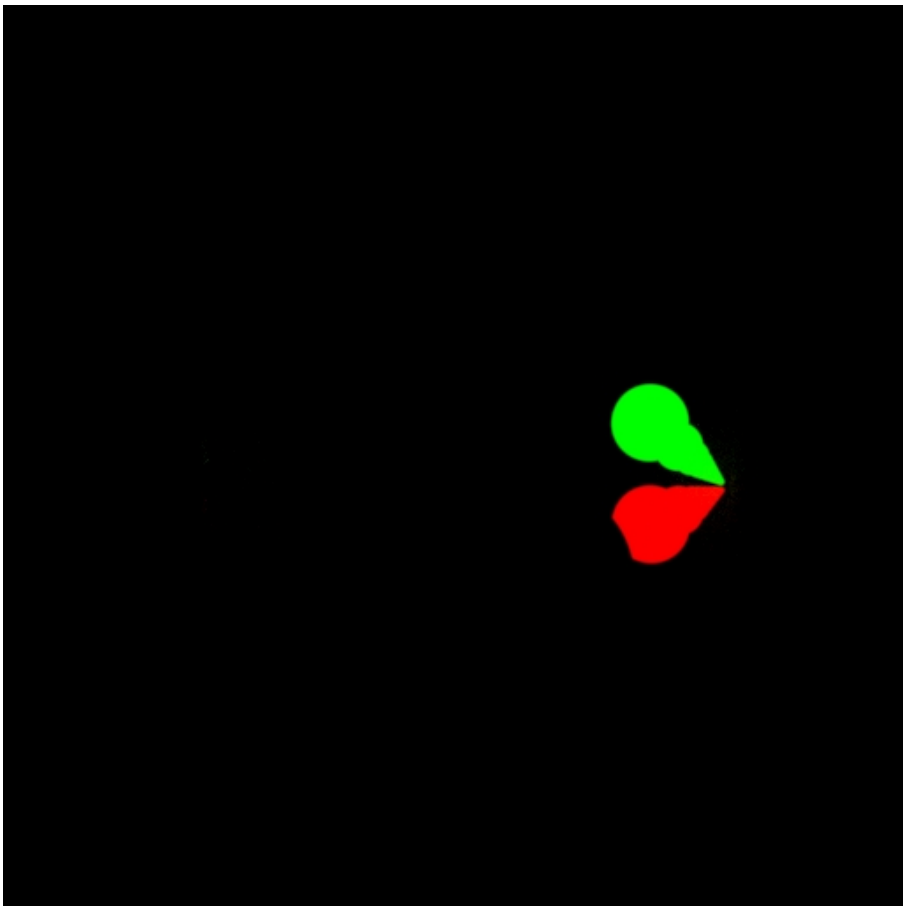


The default preset gives you this result:



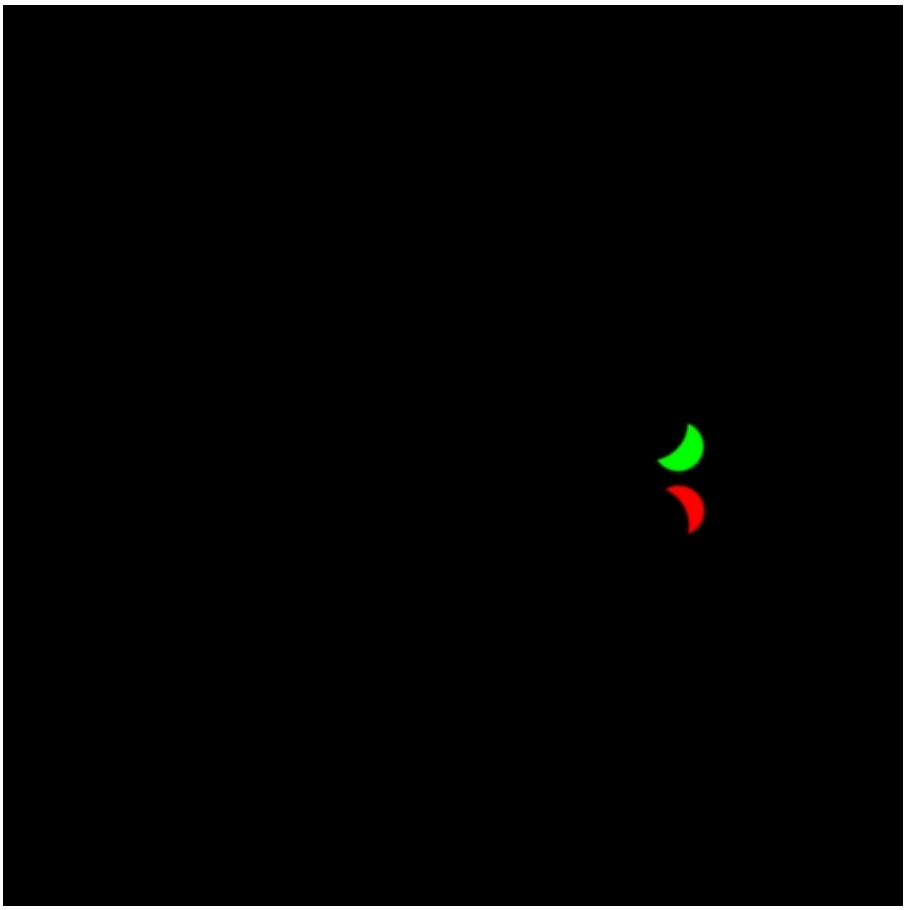
By changing this to indirect reflections you get all the spheres in the reflection:

```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;C<RS>+<.U4>[LO]
```



Or you can specify just the 3rd bounce giving a specific set of reflected spheres.

```
lpe:nothruput;noinfinitecheck;noclamp;unoccluded;overwrite;C[<RS><.U4>]{3}[LO]
```



Example File

You can download the RIB file pxrSurfaceUserColor.0001.rib
Note the Dome Light path is "C:/Program Files/Pixar/RenderManProServer-22.5/lib/RenderManAssetLibrary/EnvironmentMaps/Outdoor/GriffithObservatory.rma/GriffithObservatory.tex" and should be replaced to fit your installation path of RenderMan assets.

You can alter the DisplayChannel LPE lines at the top of the file to generate the different images on this page, some have been included by default to get you started.