Using LPE

Hand-authoring LPE can be a tedious process and for artists it's not always desired to do these manually. This document aims to improve the artist's understanding and usage of LPE.

Understanding these concepts and applying them will make you a hero to many pipelines given the power and flexibility of the feature.

In production it often becomes necessary to create non-physical effects and art-directed changes to your shot. Light Path Expressions (LPE) are a powerful way to collect and output specific light paths for alteration later.

Some examples of the power of LPE are the ability to:

- Allow for additive compositing, preserving the physical nature of the image and avoiding artifacts through multiplication or division in post processing.
- Output a specific bounce of light (2nd, 3rd, etc.) or even a range of bounces (4th to 8th bounce of light). You can use this to isolate effects or even noisy bounces for subtraction.
- Output a specific object to an LPE. While you can use Cryptomatte to isolate objects with an ID, you can also output the object or collection of objects and their contribution separately.
- Isolate a particular noisy light path for subtraction in comp or an LPE without these light paths to composite.
- Create a non-physical effect like differently colored caustics from their source, change the color cast of an object, or scale lighting.
- Output per-light AOVs for manipulation later.

Note that trace sets can also be used to accomplish some of these effects but since trace sets are global in the render, LPE will obey these trace sets (meaning data that would have been collected otherwise is restricted by the trace set).

Light Path Expressions handle light transport, as such they collect illumination by default and not shadows (shadows are simply a lack of light) A specialized Shadow LPE is provided for compositing tasks as part of our Holdout workflow. Since these follow the physics of light, your LPE will use addition (plus) and subtraction (minus) as adjustments in compositing. This avoids artifacts introduced by multiplication and division that should happen at the sample level, not on final pixels.

LPE, Cryptomatte, and Trace Sets should improve the flexibility of your pipeline and when managed correctly, avoid expensive re-rendering for artistic tweaks.

The Basics

An excellent primer with useful visuals can be found in a tutorial video from Lollipop Shaders. Begin at Class 6 for the LPE introduction.

To understand what is collected in an LPE, we will talk about the parts below and some context. Firstly, we always begin at the camera, so we begin an expression using C for Camera as the starting point for light collection.

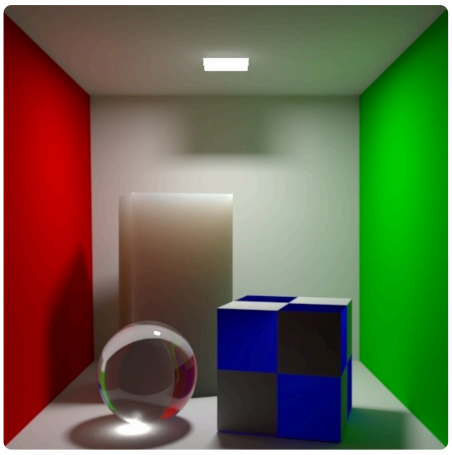
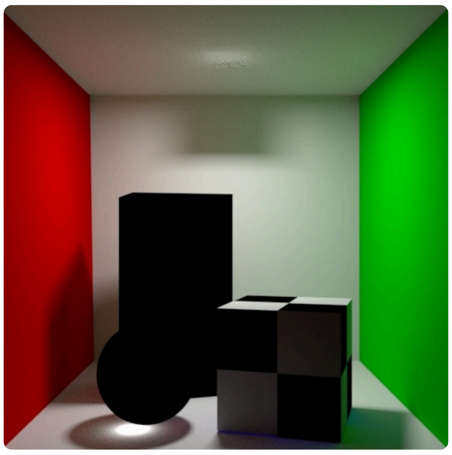
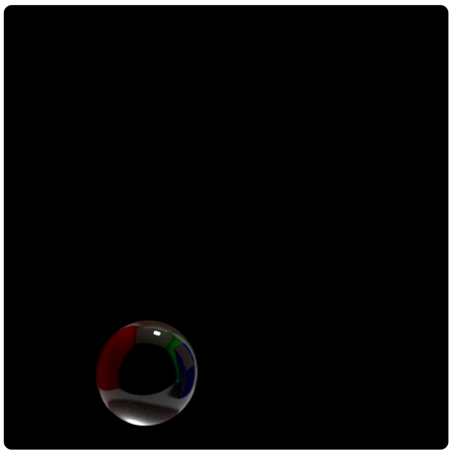
Scattering Types

Light is scattered when it strikes an object. It is either reflected off, absorbed, transmitted, or some combination of the three. In the LPE syntax we list two types of events:

- Reflected Light (*R*) is light that reflects off the object and back into the scene. This light carries information about the objects reflective properties like its diffuse color and what the object might reflect more sharply of other objects around it.
- Transmitted Light (*T*) is the light that transmits into and possibly through the object. Subsurface scattering is a very diffuse transmission of light. Refractive objects have a sharper specular transmission of light through the object.

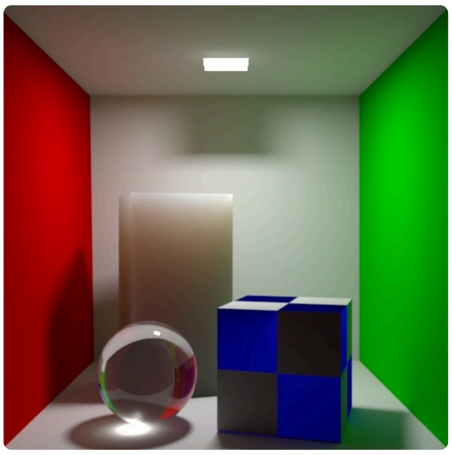
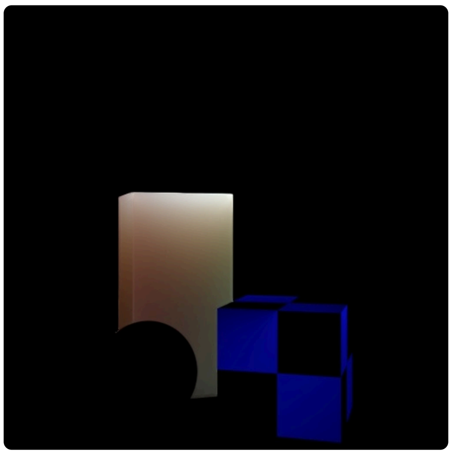
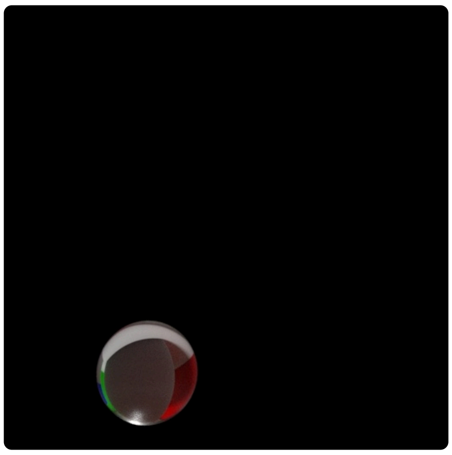
The tokens for these events are are R and T as shown above. You may specify which you want to collect or both. Later we will discuss some shorthand for specifying either but for now we'll use the tokens. Light that is absorbed is handled by the material itself and since it doesn't travel back into the scene (its energy absorbed like light in the real world) we don't collect it.

Below is a beauty render and the diffuse and specular reflection effects rendered alone in an LPE.

```
Diffuse Reflection: lpe:C<RD>*[<L.>0]
Specular Reflection: lpe:C<RS>*[<L.>0]
```

Below is a beauty render and the diffuse and specular transmission effects rendered alone in an LPE. This includes translucency and subsurface scattering.

```
Diffuse Transmission: lpe:C<TD>*[<L.>0]
Specular Transmission: lpe:C<TS>*[<L.>0]
```

Scattering Events

A scattering event is light scattered from a reflection or transmission. In RenderMan we provide two events and one special User event:

- Diffuse (*D*) scattering events are the type we associate with diffuse surfaces like paper, rough wood, skin, dust, cardboard, etc.
- Specular (*S*) events are shiny or dielectric type scattering. This covers things like shiny car paint, metals, plastics, water, glass, etc.
- User (U) events are available to shader writers interested in using them to store some other information. While mentioned here, the documentation will cover the D and S events.

You can specify a chain of events if you like, such as DDS, meaning Diffuse event to Diffuse event to a Specular in that order. The LPE would collect anything that happens in this specific order and store them. For simplicity we'll cover more common cases first.

```
Diffuse (All types, direct and indirect): lpe:C<[RT]D>*[<L.>O]
Specular (All types, direct and indirect): lpe:C<[RT]S>*[<L.>O]
```

Lights and Emissive Objects

Lastly there are the L and O tokens for lights and visible lights/emissive objects respectively. If you omit the O token you will not store emissive objects (light glowing PxrSurface materials) or visible light sources like mesh lights. Instead you would only see their effects.

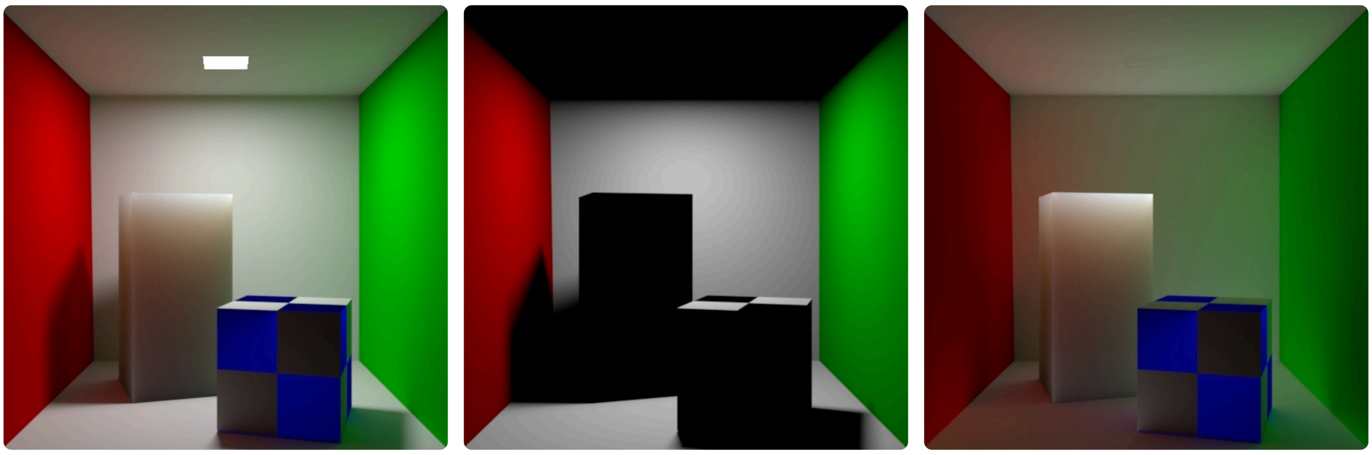
Direct and Indirect Light

In ray tracing images, we mimic what happens as light travels through a real world scene.

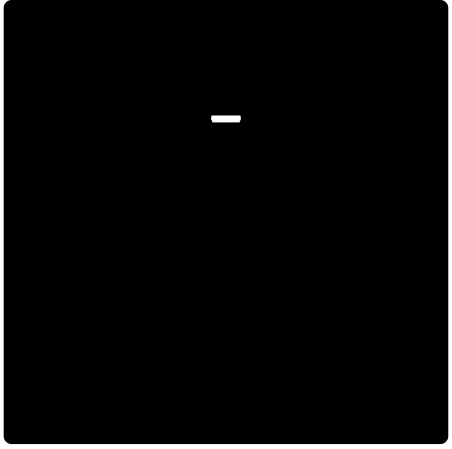
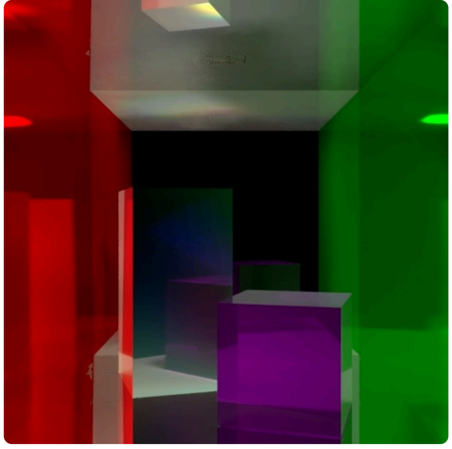
- Direct Lighting is when light from a source strikes an object directly without anything between the light and the object and seen by the camera. Imagine a flashlight on the wall, the sun on a sidewalk, or the light from this screen onto your desk. That's direct lighting.
- Indirect Lighting is when light has already interacted with other objects before reaching the camera. Imagine a mirror, the light is illuminating everything that is seen indirectly in the mirror (you're lit by a light that then bounces to the mirror and then into the camera, you're seeing the effect of lit objects indirectly in the mirror). Or a whole room can be lit up by a window where the sun is only shining on the floor. These are all indirect lighting examples. Huge amounts of information in our daily lives is indirect. Right now you may be in a room where fixtures cover the lights, that means everything you see is indirectly lit.

There is a funny thing about specular objects *without* diffuse shading, they only look like the things they reflect. If you were to render the Cornell Box scene examples with perfectly specular surfaces and a light, all you would see are reflections of the light. What does a mirror look like? Can you describe a mirror that has nothing to reflect? Or does the mirror look like what it's reflecting? This is why indirect results are so important for realism and ray tracing.

A diffuse scene using subsurface and translucent (both called diffuse transmission) effects.



A specular scene using iridescence and transmission (glass).

Now that we have some understanding of the main components, let's begin by looking at the parts used to organize an LPE

Defining Events and Types

Angle Brackets

< > are used to choose/define a single event. For example, when deciding I want a specular reflection event, I can say <RS> for a reflective specular event. I cannot create an event that is either/or, meaning I can't say <RT> because that's two different types (reflection and transmission). Instead it must be a single type and event defined in the angle brackets. You can also put these in order to collect a light path in a specific order, such as <RS ><TD > as to say "One reflective specular event and then a transmissive diffuse event next".

Square Brackets

[ ] are used to define events of any/either/or. So in the example above I could say [RT] meaning one Reflection or Transmission type and [DS] which means Diffuse or Specular event. This is pretty common when you need to collect light along a path with many different possibilities. An example might be <[RT][DS]> which means a *single event* (note the angle brackets) with either a reflective or transmissive type (first set of square brackets) of a diffuse or specular event (the second set of square brackets).

Pipe

The pipe, | means "or" and can be used to say you want either/or inside an event. This is often used in conjunction with the parenthesis explained below but there are many uses.

Parenthesis

() are used to encapsulate a series of events. This operates most like the purpose of parenthesis in a mathematical operation, grouping items to be considered together. Such an example would be to have two different sets of events you want to collect but placing a pipe, | to say "or" between these encapsulated events. C(DS)|(TD)L would be Camera to Diffuse to Specular event or Camera to Diffuse Transmission event. I can also use these for things like LPE groups to say something like ('eyeball'|'tooth') where I would store the path if it interacted with either the eyeball or the tooth groups.

Caret

The caret ^ is used to exclude something. Sometimes your light path may have an event you want to avoid. In many cases this could even be an LPE group (explained later) you wish to omit. Let's say you want all the light paths *except* for a couch in a diffuse reflection pass, C<RD[^'couch']>*[<L.>O] will return everything except for the couch.

Shorthand

Above we are explicitly stating what we want using the tokens. We can use shorthand and some following examples will use it. I can substitute a . (period) for the events and types.

Instead of writing out the tokens, just use a . for "whatever" if you need not be specific. The period is the way to shorthand an expression while other ways may simply omit a token, only the period is used in place of a token.

```
Diffuse (All types, direct and indirect): lpe:C<.D>*[<L.>O]
Specular (All types, direct and indirect): lpe:C<.S>*[<L.>O]
```

You could even shorten the beauty AOV by saying:

```
A longer beauty LPE can be seen as
Beauty: lpe:C<[RT][DS]>.*[<L.>O]

Can be written as a shorter version below (notice the omission before the asterisk meani
Beauty: lpe:C<..>*[LO]
```

This can be shrunk even further (you notice we remove a couple things where "any" or the . just gets omitted) but we'll go over what the LPE parser assumes later. For now we'll keep it tidy and readable.

Defining Bounces

Since we're tracing through the scene collecting light, it might be important to specify how much light you need or want to collect. This is usually meant to separate the direct and indirect events mentioned above. But you may also specify how many bounces along this path you want to collect rather than let the renderer collect all of them. Collecting specific bounces is covered later.

To begin we'll look at the usual notation for Direct and then Indirect light collection.

Above we were defining events and their type. But we need a way to say how many of these we want. We could manually string together events in a particular order but that becomes a painful experience in copy and paste. It's better to let the renderer make the decision for you. We have two common ways to define how many of our defined events we want.

Asterisk

* means zero or more of these preceding events. Since it's zero or more, it will collect a direct lighting event and the ones that follow it (meaning indirect) til they reach a light source. It's the simplest way to capture light for the whole path. The following collects only diffuse reflection and all the events that match diffuse reflection until it reaches a light.

```
lpe:C<RD>*[<L.>O]
```

Plus

+ means one or more events. This means after the initial event, begin collecting others. This means it will collect indirect lighting only as the first (direct) event is skipped. Below is only indirect diffuse reflection (the typical indirect light AOV)

```
lpe:C<RD>.+[<L.>O]
```

None

If none of the notations are used, it collects an event and then goes to the light, collecting only direct lighting. Below we omit the * and the +, only direct diffuse reflection is collected.

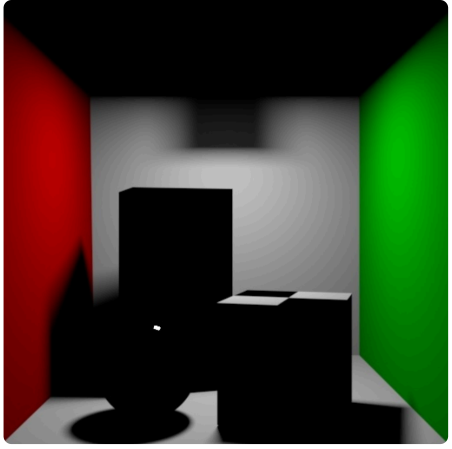
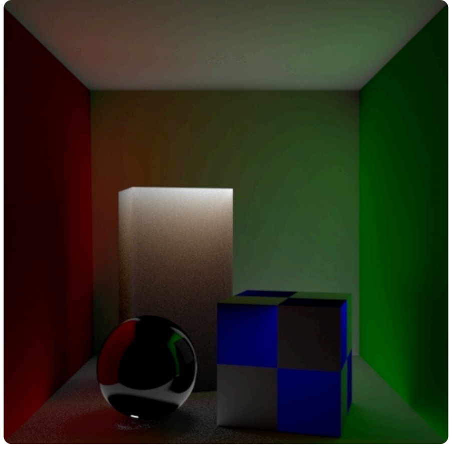
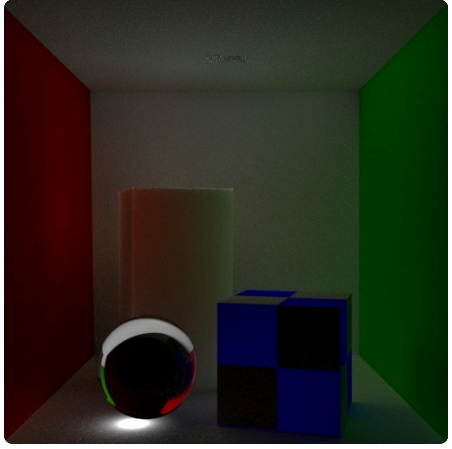
```
lpe:C<RD>[<L.>O]
```

Specified Bounces

Above we use the most common ways to collect light paths, you either want all, direct, or indirect. This handles most scenarios you come across. But there may be reason to do something with a specific bounce or set of bounces. This can be anything from non-physical scaling of the light (brighter or darker) or isolating a specific bounce where noise is introduced and subtracting it from the final result. This last example is best when you can isolate the object(s) causing the noise. This will be covered under LPE Groups. Below is an example of rendering only the first (direct light), second, and third light bounce. These LPE are all very similar to the Beauty LPE but we exchange the * for all bounces with a specific bounce contained in a curly bracket {} For example, {1} collects the first bounce or the direct light frm the ceiling rect light to the object and then immediately to the camera. The second bounce is {2} which says the light must interact with something one more time before being stored, this is the first indirect bounce after light meets a surface. This goes on until the path length is terminated by the parameters in the Render Settings, Max Specular/Diffuse depth and/or Max Path Length for the integrator and possibly Russian Roulette early termination (all the energy was already absorbed leaving you with no more information to store).

```
lpe:C<[RT][DS]>{1}[<L.>O]
lpe:C<[RT][DS]>{2}[<L.>O]
lpe:C<[RT][DS]>{3}[<L.>O]


Shorthand:
lpe:C<..>{1}[LO]
lpe:C<..>{2}[LO]
lpe:C<..>{3}[LO]
```

The curly brackets can also take a range of values to render a set of bounces. Maybe you're only interested in collecting the effects of the 4th through 8th bounce? Or maybe you know where to begin or end but want the renderer to decide where that is.

Think of it like this, A is where we will begin and B is where we will end. It's written like this: {A,B}

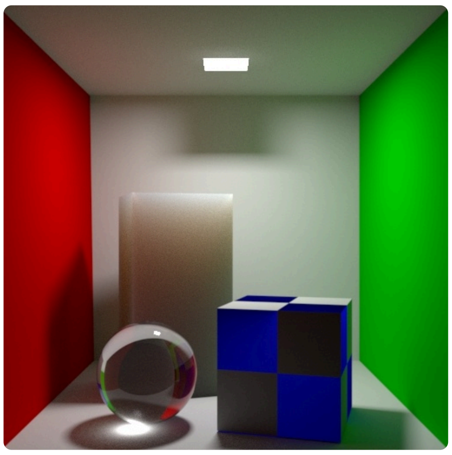
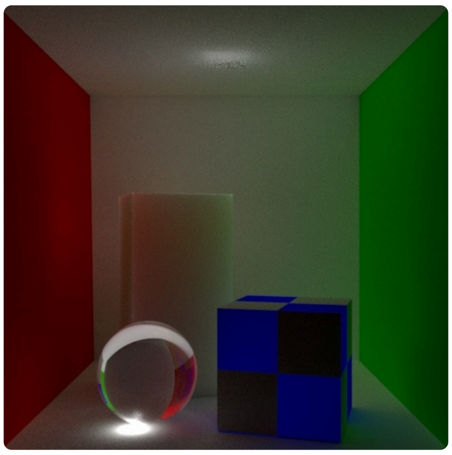
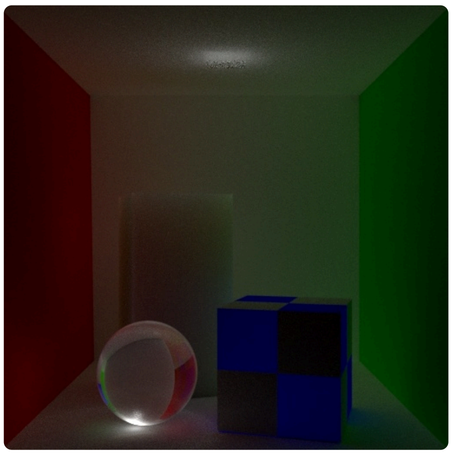
```
Collect just the 4th through 8th bounce
lpe:C<[RT][DS]>{4,8}[<L.>O]

Collect all the bounces until you reach the 5th bounce
lpe:C<[RT][DS]>{,5}[<L.>O]

Start at the 3rd bounce and go until the path is finished
lpe:C<[RT][DS]>{3,}[<L.>O]


Shorthand:
lpe:C<..>{4,8}[LO]
lpe:C<..>{,5}[LO]
lpe:C<..>{3,}[LO]
```

The following examples show these actions. Note that when we leave the beginning or ending blank, the renderer makes that decision on where to begin or end the collection.

The Beauty Render

Let's look at an expanded beauty LPE. And by "expanded" I mean with all the parts spelled out and not in shorthand.

```
lpe:C<[RT][DS]>*[<L.>O]
```

Camera to <[reflection OR transmission] OF [diffuse OR specular]> ALL light bounces TO [lights OR mesh/glow objects]

You can now understand each of the parts above and how specifying the right tokens (or even simple swapping them out) can give you the results you need for compositing a frame after making adjustments.

The above LPE can be used to output not only the beauty, but can be used to output something like the beauty of a particular light group, this is a very common "ask" by artists. Below we'll dig deeper into more advanced workflows.

Per Light LPE

For artists rendering complex stills, it might be important to separate lights and light groups to change contribution later instead of re-rendering or creating many wedge tests. For animation, per-light AOVs are much less common as complex VFX composites already have many layers on top of AOVs for those layers. Splitting this further into selected light passes can further multiply the number of passes a compositor will handle to adjust the beauty. This is possible and should your pipeline handle this extra set of data, LPE can provide the control you need.

In RenderMan for Maya we handle this using the lights drop down menu in the AOVs tab where you define your LPE Display Channels. But other software may require you manually insert the light groups.

First, you must specify which lights belong to which groups. This is done at the light shader level. You notice a string (text) field in the light parameters for an LPE Group. This is where you define your groups in plain text. For example, in the classic three-point lighting setup you have three lights or groups: key, fill, and rim lighting. You can specify this in the light shaders and then call these groups using the LPE syntax of a name in single quotes, like 'key' or 'fill'.

```
lpe:C<[RT][DS]>*[<L.'key'>O]


```

```
Shorthand
lpe:C<..>*[<L.'key'>0]
```

The above LPE will render a beauty of the scene lit by the lights tagged as "key". Replacing 'key' with another group would render the other group as specified like 'rim' or maybe you need the HDRI as 'environment' separated into an AOV.

If you needed more than a single group, you could use the pipe to say "or" another group or groups.

```
lpe:C<[RT][DS]>*[<L.('key'|'fill')>0]


Shorthand
lpe:C<..>*[<L.('key'|'fill')>0]
```

Of course you can also ask for other data from a path to a light group like indirect diffuse reflection (usually known as indirect lighting) for later scaling in compositing.

```
lpe:C<RD>.+[<L.('key'|'fill')>0]
```

Now you can output whatever you need based on light groups!

Per Object LPE

You can also render specific objects and effects. This is similar to the light groups but this is often found on a transform of an object, this LPE group tags these objects or collections for separation.

Imagine you have a particular thing you're trying to capture, you can do that with an LPE. You notice that in all the LPE we've been writing it looks like it can be done in 3 parts:

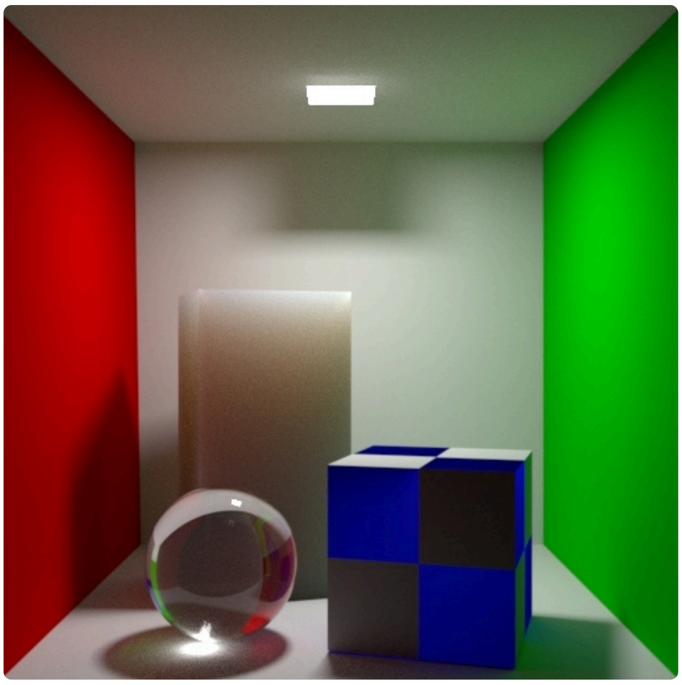
```
[What I'm Rendering] [Light Bounce Paths] [Lights I'm Using]
```

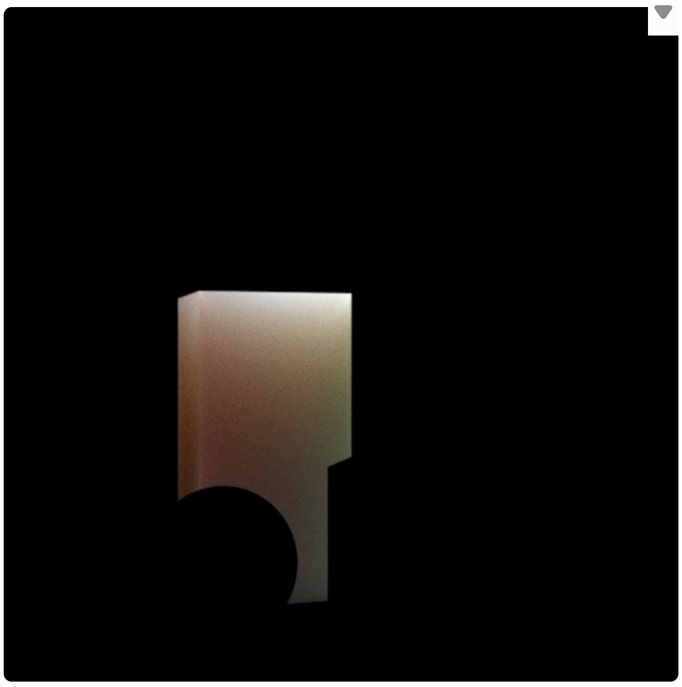
So now let's render something specific. In the Cornell Box example we have a tall subsurface scatter rectangle in the back. I'll render it by itself.

```
lpe:C<[RT][DS]'scatterCube'>.*[<L.>0]
```

```
Shorthand
```

What I'm rendering is the tall rectangle and all the light that lands on it.



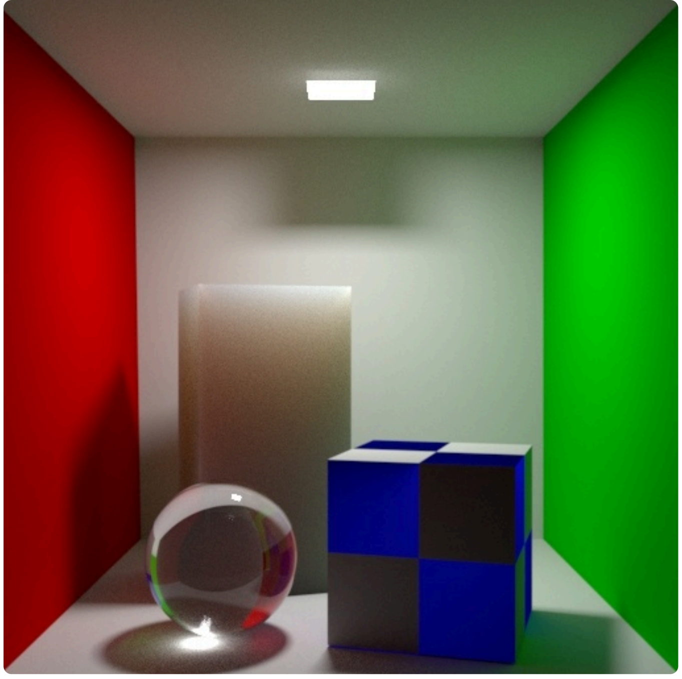
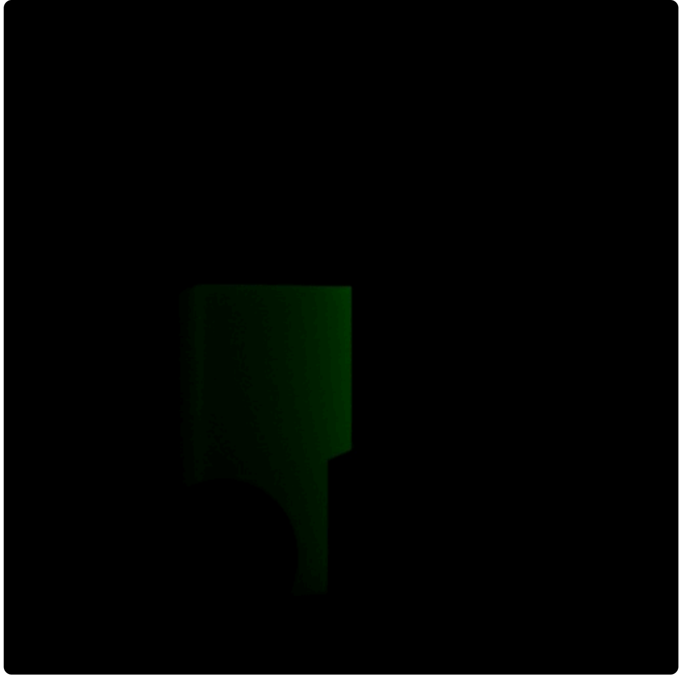


Now, let's render it and only the influence of light off the green wall to the right.

```
lpe:C<[RT][DS]'scatterCube'>+<[RT][DS]'rightWall'>[<L.>0]


Shorthand
lpe:C<..'scatterCube'>+<..'rightWall'>[<L.>0]
```

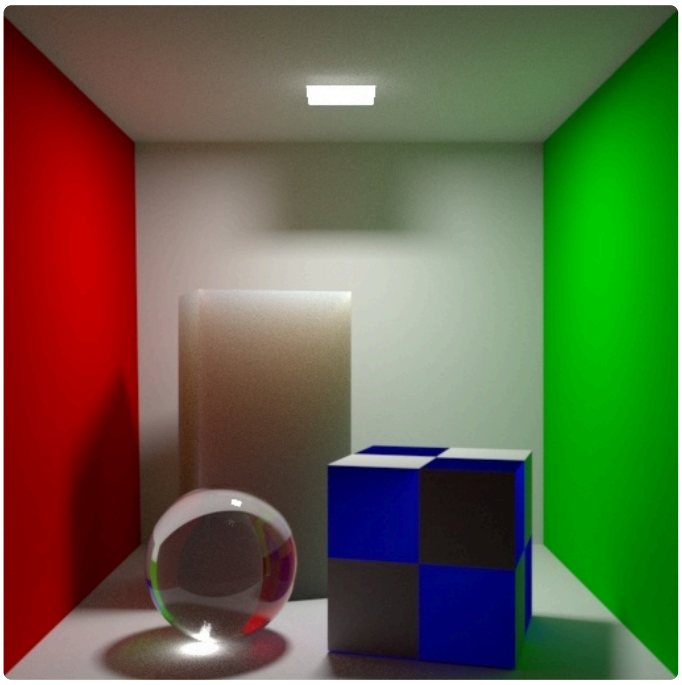
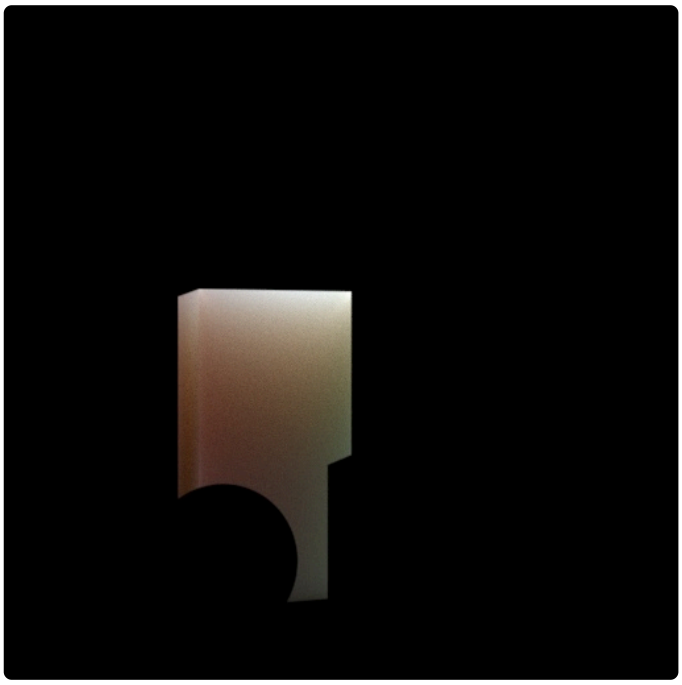
So we're rendering the same cube but this time the light path has to exclusively include all the light bouncing off of and through the rightWall object. So our result is a very green object since we captured just the green wall's bounce light. You can see now how to separate light paths to get what you need to adjust for later compositing. I'm rendering the tall rectangle and all the light that lands on it *after* leaving the green wall.

Of course I can also exclude just the green wall. Maybe the green wall is generating noise, I can avoid those light paths by using the caret and a simple tweak of my LPE

```
lpe:C<[RT][DS]'scatterCube'>+<[RT][DS][^'rightWall']>[<L.>O]


Shorthand
lpe:C<..'scatterCube'>+<..[^'rightWall']>[<L.>O]
```

The Importance of Naming

Naming your channels is important not just for pipeline reasons and readability, but because the Denoiser utility requires that you follow a naming convention in order to be denoised. Since the tool isn't psychic, it needs a hint as to what is inside the framebuffer/AOV and it uses the name to decide this. There are two main ways the Denoiser filters results, and the way triggered by a "diffuse" keyword is that the albedo is divided out before denoising happens, this is done to prevent texture smearing. So when naming your AOVs, keep that in mind knowing that you'll most likely want your results to be denoised. Below are the acceptable prefixes given again.

The pass images can be named anything else, *but must have color channels named/prefixed as one of:*

- `diffuse`
- `specular`
- `directdiffuse`
- `directspecular`
- `directDiffuse`
- `directSpecular`
- `indirectdiffuse`
- `indirectspecular`
- `indirectDiffuse`
- `indirectSpecular`

LPE Prefixes Explained

You may see other areas where another prefix is added to the LPE. This is typically for special handling and data.

- `unoccluded` – returns unoccluded or unshadowed result. Using this before your LPE will result in no shadowing.
- `noclamp` – returns unclamped result.
- `nothruput` – does not apply thruput (throughput is the accumulative albedo of the objects hit by rays). Your results could be attenuated based on throughput of the light path. Using this means the full value is retained, useful for masks.
- `shadows` – returns collected shadows. This relies on the shadow sample filters and additional buffers from the Holdout Workflow.
- `holdout`s – returns only holdout light paths (light paths with one or more holdout events)
- `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. Otherwise your values may continue to accumulate light and be incorrect, for example, a pure red of 1,0,0 can become 6,0,0 and be incorrect.
- `noinfinitecheck` – do not do any infinite check.

User Lobes

By default you've been seeing us use Diffuse and Specular lobes, but a User lobe can be built into a material for data purposes. These lobes are evaluated with the material but not output to a beauty as a lighting result. These lobes must be specified in an LPE to work, shorthand assumes only Diffuse or Specular substitutions. A useful example are the User Color and Position lobes of PxrSurface.