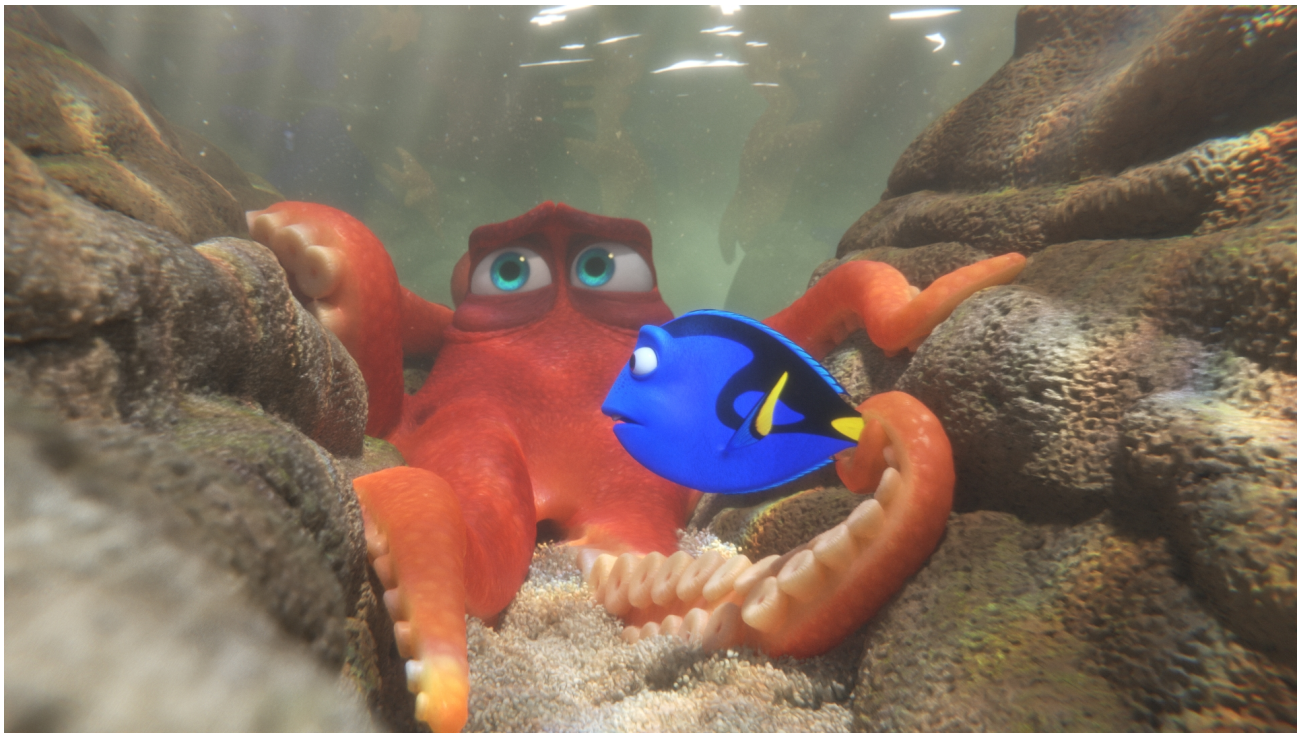


Finding Dory ©Disney/Pixar 2016.

Towards Bidirectional Path Tracing at Pixar

Christophe Hery, Ryusuke Villemin, Florian Hecht

1 Introduction

On *Finding Dory* and *Piper*, we were faced with rendering a lot of water, or creatures and sets seen through water and glass. We worked with production artists to give controls for resolving these difficult light transports. We also learned to cheat these effects where appropriate. In these course notes, we will elaborate on some of the constraints that bidirectional path tracing imposed on us, and our solutions for them.

2 The idea behind Light Path Expressions

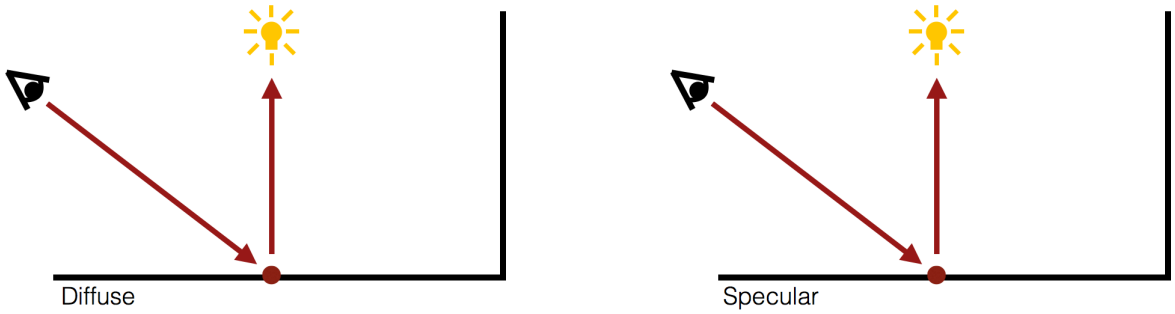
Light Path Expressions (LPEs) were introduced to the graphics community by Heckbert [Hec90] to describe rendering algorithms. They were more recently used in Open Shading Language [Gri] as a generalization of Arbitrary Output Variables (AOVs) in some core ray-tracing computation engines, among them RenderMan RIS. During the production of *Finding Dory*, we found that we could employ this technology for more than AOVs. We are going to give some examples, after reviewing their syntax.

LPEs are phrases made out of a very simple vocabulary. Words are:

- **C**: camera
- **D**: diffuse (or **D2** for diffuse lobe 2, **D3** for diffuse lobe 3, etc.)
- **S**: specular (or **S2** for specular lobe 2, **S3** for specular lobe 3, etc.)
- **L**: light
- **O**: emissive object.

With these, one can construct simple expressions, such as:

- **CDL** for a path going through the camera, hitting a diffuse object, then joining to the light(s): see Figure 1a
- **CSL** would be similar, with a specular object: see Figure 1b.

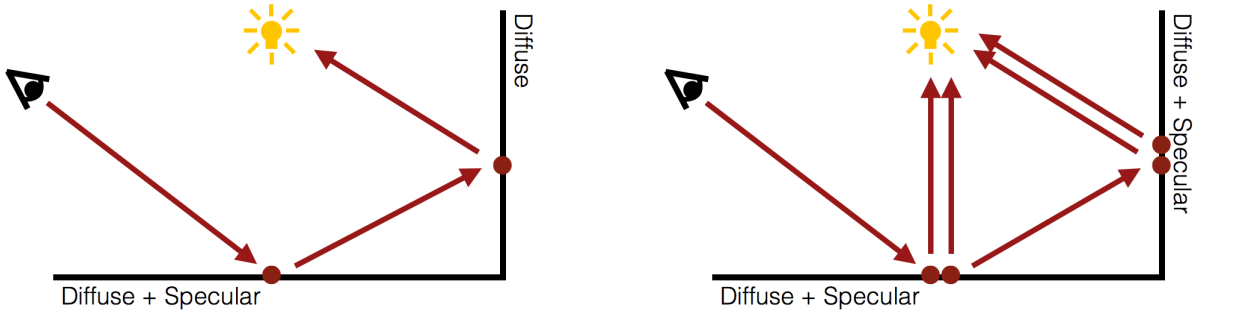


(a) **CDL** (b) **CSL**

Figure 1: Simple LPEs.

The grammar adds some extra selections with < > and [ ] tokens, as well as the special * and + signs:

- < > specifies the event type (**T** for transmission and **R** for reflection), as in **C<TS>O**, would mean paths reaching emissive objects after a refraction event
- [ ] is a way to say "or", as in **C[DS]DL**, see Figure 2a
- * means 0 or more events, + means 1 or more, so **CDS+L** would be caustics, while **CDS*L** would be direct diffuse and caustics.



(a) **C[DS]DL = CDDL or CSDL** (b) **CDL and CSL and ((CDDL and CDSL) or (CSDL and CSSL))**

Figure 2: More Complex LPEs.

Figure 2b requires some additional explanation. In [VH15], we briefly presented a technique to efficiently sample between all of the lobes of our uber BxDF. This "multi-lobe" MIS gives a lot of benefits for direct lighting, but tends to complicate the logic when tracking LPEs in indirect paths. So we enable it only in direct illumination. Thus when picking a direction for BxDF sampling for indirect, we choose one of the lobes (and do not evaluate the others). Hence the "or" in the expression. From there, we want to sum up ("and") the direct and indirect contributions. In the end, this full path description is not really a valid LPE syntax. It contains many sub-LPEs, and is meant here as an illustration of the complexity of the paths we have during rendering.

3 LPEs for denoising

For the purposes of post denoising renders, it is desirable to output separate utility channels, for each pixel in the images, such as:

- position
- identifier
- normal
- albedo
- diffuse
- specular.

A diagram for this setup is given in Figure 3.

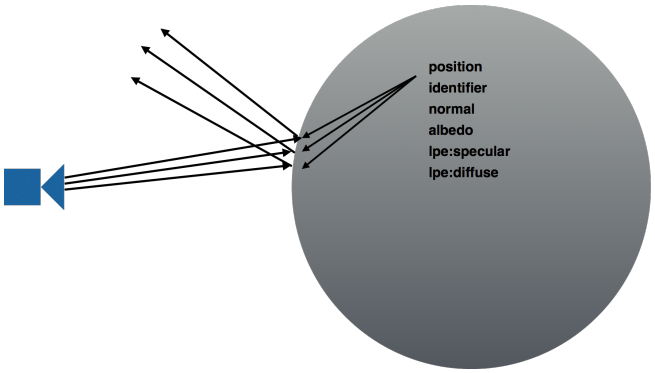


Figure 3: Simple denoising setup.

For this, and taking the case of the diffuse component, a very simple expression will extract what is needed: **CD[DS]*[OL]**. Note that this captures all of the potential indirect bounces as well as the direct diffuse illumination.

Unfortunately, on *Finding Dory*, we found that we often had elements to denoise that were behind water or glass. See Figure 4. In this case, we could mark our objects through which we would want to denoise, as "pass-through", with a special named **S2** for specular lobe 2.

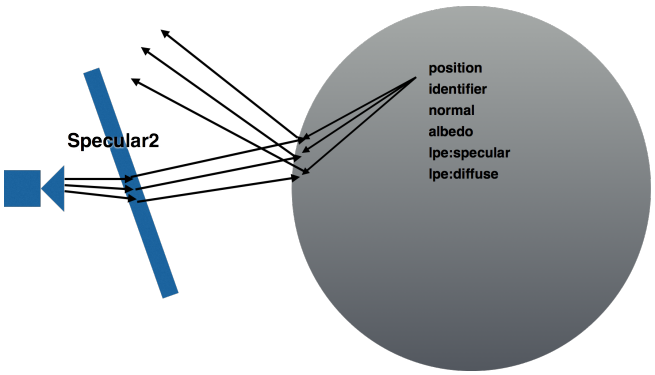


Figure 4: Behind Glass Setup. Specular 2 is our "pass-through".

The LPE for the diffuse output becomes slightly more complex: **C<TS2>*D[DS]*[OL]**. For specular, we would use: **C<TS2>*(S1[SD]*[OL])|([OL])**. Note the use of the special S1 notation, which refers to specular lobes not tagged "pass-through".

Examples of these outputs are given in Figure 6 for the final render in Figure 5.

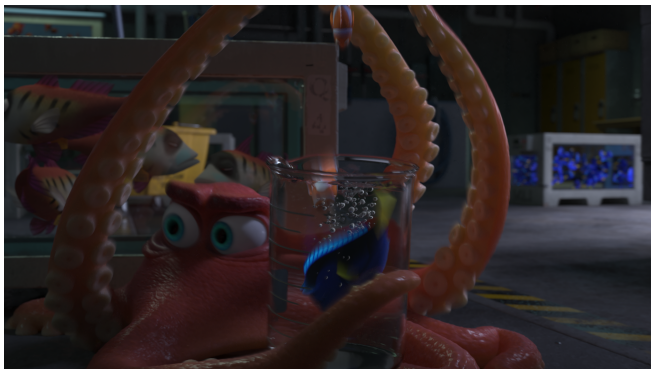
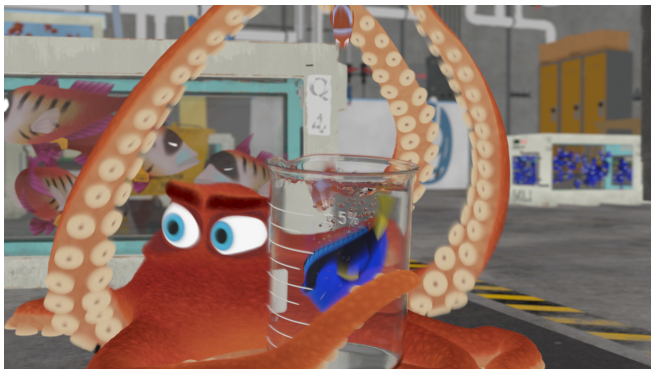


Figure 5: Beauty.

(a) Diffuse

(b) Specular



(c) Albedo

(d) Normal

Figure 6: Denoising outputs (with glass and water marked as “pass-through”).

4 LPEs for global illumination artistic control

At Pixar, artists want fine control over the impact of their lights. They often define regions of space, with ellipsoids—internally called *Rods*—where they want to manipulate the effect of the illumination (for example the intensity, or maybe saturation). With LPEs, we can let them do this editing even on indirect bounces.

See Figure 7 for the setup on a diffuse bounce that one wants to attenuate in a specific region.

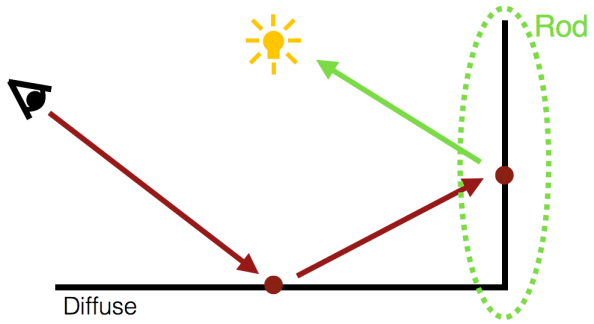
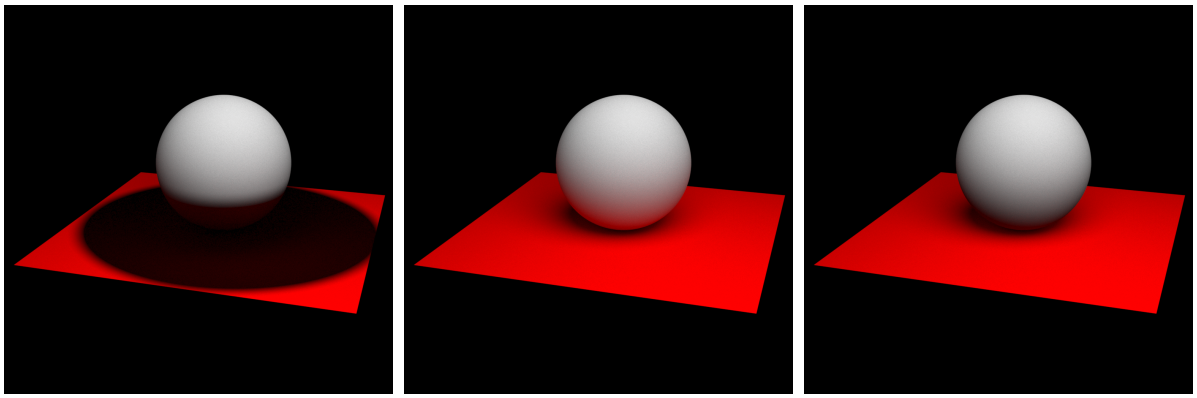


Figure 7: **CDL** manipulation in a region.

Such an operation is specified with a special light filter, that takes spatial Rod parameters and an LPE as inputs. Then, during path integration, we apply the multiplicative effect inside of the Rod region only if the path from the camera matches the LPE. This way the influence of the Rod is restricted to indirect effects selected by the lighters.

The result of this manipulation is shown in Figure 8.



(a) Region (b) No effect (c) With effect

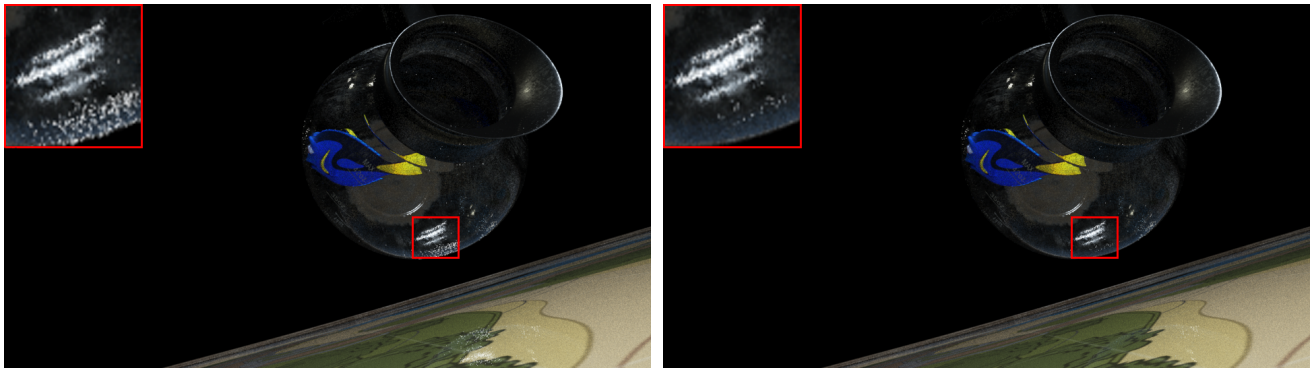
Figure 8: Applying a Rod filter. See how the red diffuse bounce from the floor is reduced within the region.

5 LPEs for firefly removal

We can get fancier with LPEs. We can specify explicit lights (or group of lights) and objects (or group of objects) for which we want to capture and edit some events. For instance, we can eliminate some highlights through glass, via this semi-arcane expression:

CS+<.S‘potglass’>[SD]+<L.‘potgroup’>.

Note the use of the ‘potgroup’ of lights and of the ‘potglass’ geometry. [1] Results are shown below in Figure 9.



(a) Not applied (b) Applied

Figure 9: Firefly removal on glass or with light shining through glass.

Hairs proved to be difficult geometries, for which tiny tracing bias issues would create undesirable and hard to resolve fireflies. To remedy this, we eliminated some of these bad paths, via expressions such as:

C[<.D‘hair’><.S‘hair’>][SD]+L.

You can see the effect of this in Figure 10.

(a) Not applied (b) Applied

Figure 10: Firefly removal on hair.

[1] Here, we also introduce a new notation: “**.**”, a wildcard that can represent any scattering event or type. For instance, **C<.S>[OL]** means the union of **C<RS>[OL]** with **C<TS>[OL]**, which can in turn be simplified into **CS[OL]**.

Finding Dory ©Disney/Pixar 2016.

6 Path Tracing

6.1 Introduction

With bigger render farms, and increasing computational power, the world moved to pathtraced rendering. The advantages are obvious and numerous: progressive rendering, billions of geometry instances, convincing physically based results out of the box, etc. However, because of the way path tracing works, it also came with a new set of restrictions on shading and lighting. To understand where those restrictions come from and what can be done to overcome them, we will first present an overview of ray tracing and Multiple Importance Sampling (MIS), and then focus on refractive objects shading.

6.2 Unidirectional path tracing with next event estimation

What we call path tracing is usually forward ray tracing with next event estimation. The goal is to solve the rendering equation [Kaj86] using Monte Carlo sampling:

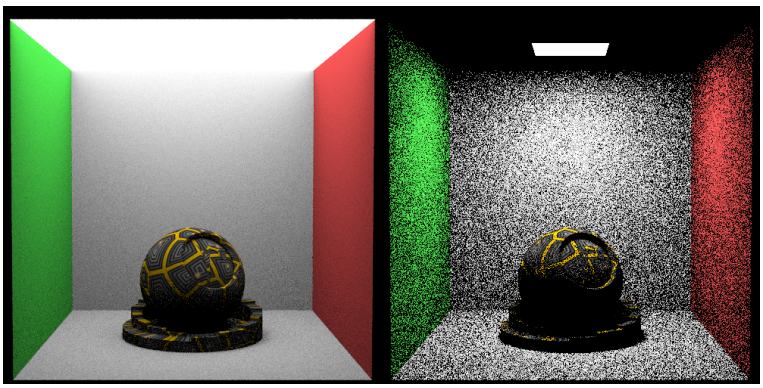
$$L_o(x,\omega) = L_e(x,\omega) + \int_\Omega f_r(x,\omega',\omega) L_i(x,\omega')(\omega' \cdot n) d\omega'. \tag{1}$$

L_o	Outgoing radiance
L_i	Incoming radiance
L_e	Emitted radiance
f_r	BRDF
x	Position on surface
n	Normal at x
ω	Outgoing direction
ω'	Incoming direction
Ω	Hemisphere of directions

Table 1: Notation.

As we can see the equation recursively describes light paths that are bouncing in the scene, ultimately connecting the lights to the camera. At each bounce, there is an interaction with the surface, described by the

Bidirectional Reflectance Distribution Function (BRDF) f_r (let's ignore transmission and other volumetric, subsurface effects in this course). While it is difficult (impossible?) to directly importance sample this integral, we can see that we can locally importance sample each bounce. Doing so gives us the simplest form of recursive ray tracing: just importance sample f_r. Although this is a simplistic algorithm, it works pretty well for diffuse objects and large light sources.



(a) (b)

Figure 11: Comparison of large (a) and small (b) light source.

But as the light source becomes small, there is less chance to hit it, and this translates into additional variance in the image (Figure 11).
This is where next event estimation comes into play. In the scene we tag special objects as lights, and the renderer is made aware of those. The difference is that now the renderer knows that those objects emit a lot of energy, so they should be sampled directly. Doing light sampling, we get to the same result (Figure 12) because the solved problem (the rendering equation) is the same, but it converges much faster.

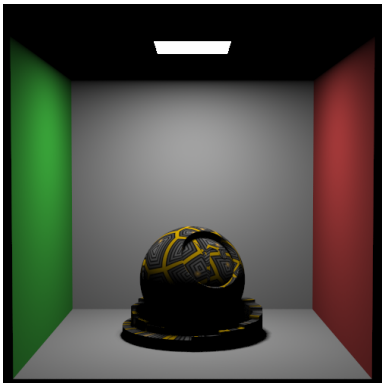


Figure 12: Small light source with light sampling.

We now have an additional problem though: we have two methods to compute the same equation (Figure 13). If we just add both results, we will end up with twice the energy (and, at most, cut the noise in half). One approach that has been taken in the past is to let the user decide to do one or the other per object. For example, just BRDF sampling for specular objects and only light sampling for diffuse objects. But this is not practical and does not account for all of the possibilities and combinations.

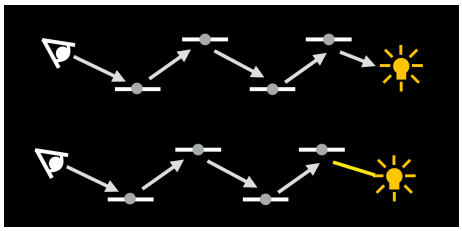


Figure 13: Top: BRDF sampling, Bottom: light sampling.

6.3 MIS

The solution to this problem, proposed by Veach and Guibas [VG95], is very simple: use the balance heuristic, based on the Probability Distribution Function (PDF) of each technique, to weigh their contributions. The PDF represents the sample distribution, i.e. the samples are going to be concentrated where the PDF value is high. This will "switch" automatically to the right technique, by favoring the one with the higher PDF.
Rough BRDFs and small lights mainly use the light sampling results, whereas specular BRDFs and large lights mainly use the BRDF sampling results. Everything in between will automatically use a mix of the two. For M sampling techniques and N_j samples per technique, we have

$$F = \frac{1}{N} \sum_{j=0}^{M-1} \sum_{i=0}^{N_j} \omega_j(y_i) \frac{f(y_i)}{p_j(y_i)} \tag{2}$$

with

$$\omega_i(y) = \frac{p_i(y)}{\sum_{j=0}^{M-1} p_j(y)}. \tag{3}$$

When using direct lighting MIS with unidirectional path tracing, we get two techniques per bounce ($M = 2$), with p_0 being light sampling and p_1 BRDF sampling.
This works well (and was our main rendering scheme at Pixar since *Monsters University*—we refer the reader to [HV13]), but still has limitations. The most notable failure cases are scenes with strong indirect lighting and scenes involving many caustic paths.

6.4 Bidirectional path tracing

Bidirectional path tracing is a generalization of the previous technique. When doing light sampling, we stopped as soon as we had a sampled point on the light source. But if we use this point to start light paths, then we will get additional sampling methods to sample paths between the camera and the lights by combining these light paths with the usual camera paths (Figure 14).

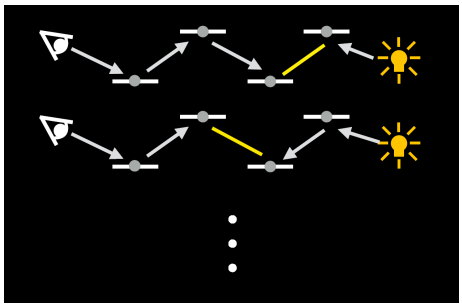
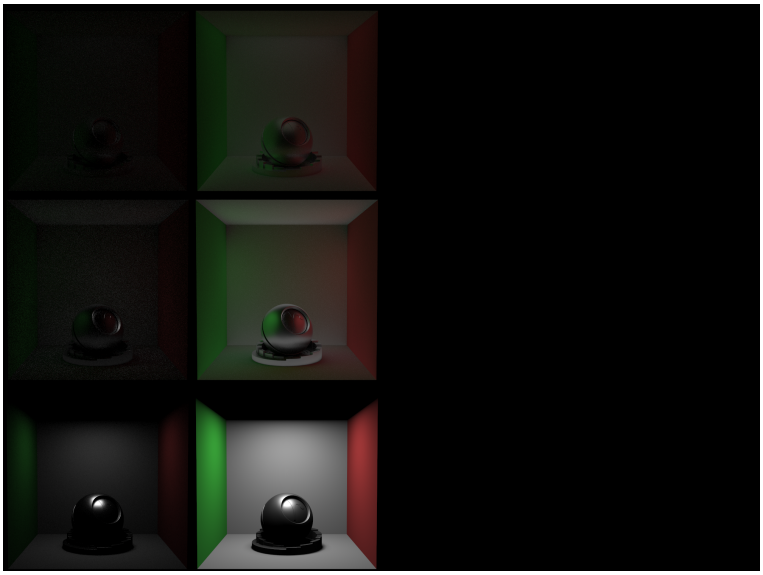
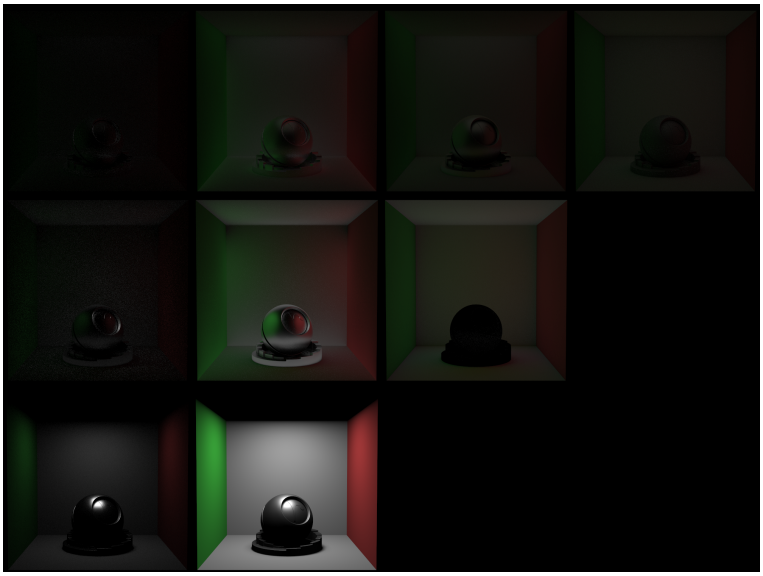


Figure 14: Bidirectional path sampling.

By using the balance heuristic, we can still combine the results of all of the methods (Figure 15). Note that the first column catches most of the specular highlights, since BRDF sampling will get a high MIS weight due to the BRDF PDF. And the second column catches most of the diffuse illumination, since, for diffuse objects, light sampling usually gives the best results. For bidirectional renders, we get additional columns that represent light paths with more than one vertex.



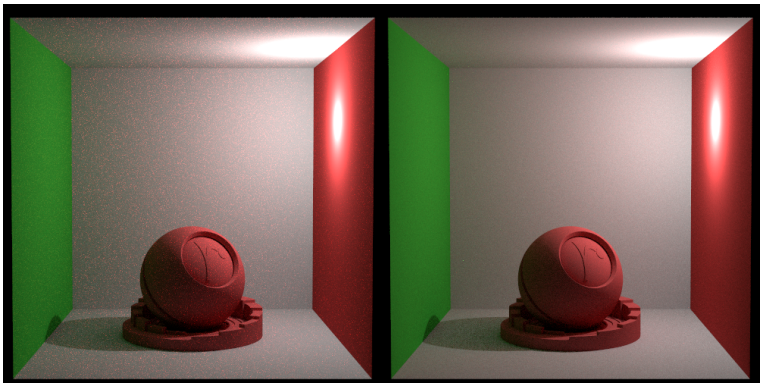
(a) Each row is a depth in the path, first column is the BSDF sampling, second column shows the light sampling



(b) For bidirectional renders, as we go deeper we find additional sampling techniques

Figure 15: MIS weighted contributions.

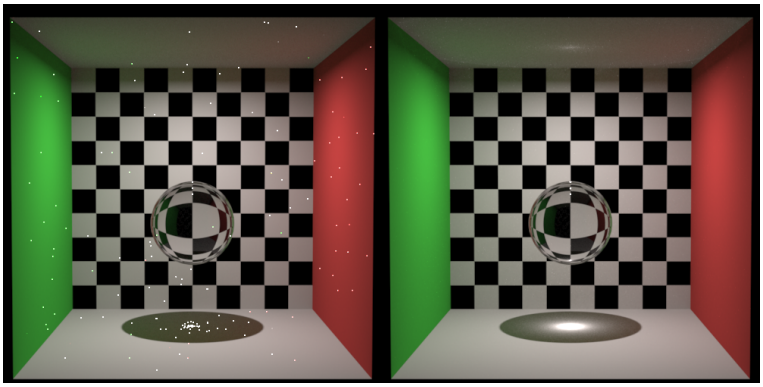
In the case of strong indirect lighting, unidirectional path tracing does not perform well. This is because the first bounce is as bright as the light, but since it's not a light the renderer won't directly sample it. With bidirectional path tracing, the first bounce becomes the second vertex in the light path, so it will get importance sampled. All of the red fireflies due to the first strong bounce from the red wall are gone in the bidirectional render (Figure 16).



(a) unidirectional (b) bidirectional

Figure 16: Strong indirect lighting comparison.

Caustics paths, which are made of specular reflection and transmission, are a second common problematic case. The described path-tracing techniques can only render those paths when the light path ends on a diffuse object and we make a connection directly to the camera (Figure 17). For more complex cases, such as caustics seen through reflection, we need to use photons. This is outside of the scope of this course but there are now ways to multiple importance sample biased and unbiased methods: unified path sampling (UPS) [HPJ12]; vertex connection and merging (VCM) [Geo+12]. Without employing complex integrators, if photons are used only in very particular scenarios, then we would mix them back to the final result by hand, disabling any paths left to photon sampling in unidirectional path tracing, and then adding back the photon map contribution.



(a) unidirectional (b) bidirectional

Figure 17: Caustics comparison.

6.5 Shading requirements

Now we are aware that to render efficiently we need to be able to sample using multiple techniques. For BRDFs, it means that:

- BRDFs must be reciprocal, otherwise we will get different results based on the sampling method we use.
- BRDFs must be energy conserving. To correctly render complex effects, we need a high number of bounces. If we were to gain energy at each bounce, the result will diverge.

- We need to be able to evaluate a BRDF (of course!), but we also need to sample it efficiently. A fancy BRDF model without good importance sampling is impractical!

And of course, if we want a consistent system, we need the same kind of requirements on the lights:

- Lights must be energy based; we need to be able to not only evaluate but also sample them.
- This is not a requirement per se, but lights that can have an optimized sampling method based on the shading point are preferred. For example, a sphere light should not sample a point on the hidden hemisphere of the shading point, and a rect light should sample based on the apparent angle [PH14].

7 Refractive materials

Finding Dory ©Disney/Pixar 2016.

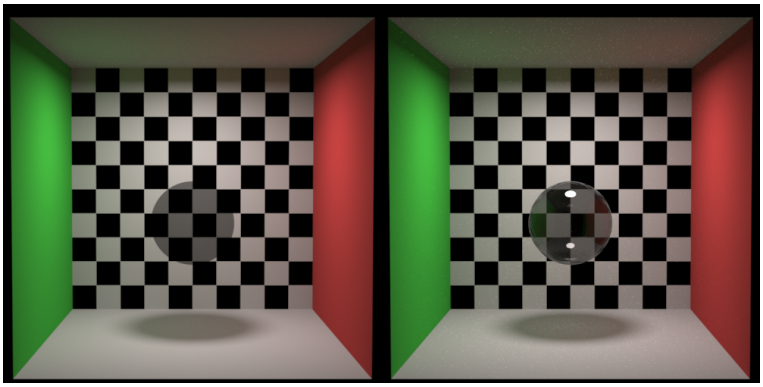
7.1 Description

Let's now focus on refractive object shading. Usually those objects are a difficult case because they are a mix of lighting and shading. Really what we see when looking at a refractive object is the lighting through multiple and complex reflections and refractions. Lighting and shading both need to be done right to get a correct image.

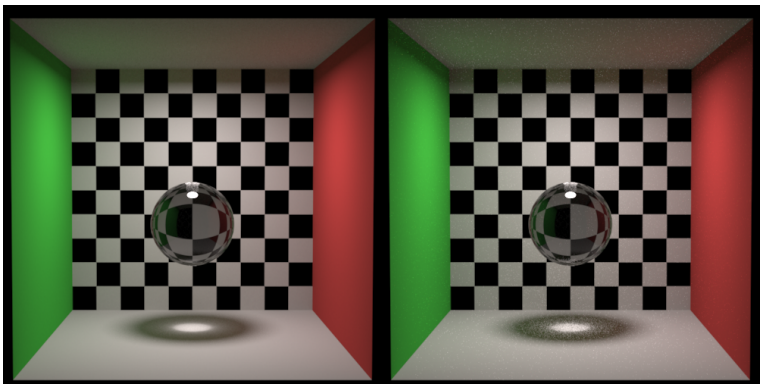
7.2 Refraction types

Even before having bidirectional path tracing, people were rendering refractive objects by making certain assumptions and simplifications.

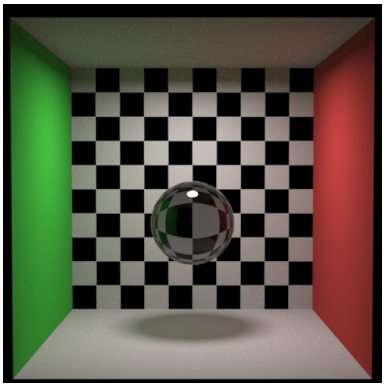
- Ignoring the light bending due to the index of refraction (IOR), refraction would simply become transparency, and rendering transparent objects is just coloring the light when passing through the object (Figure 18a). This is an over simplification and does not make the object feel refractive.
- A simple improvement can be done by adding reflection and taking into account the Fresnel effect on it, so that you get more reflection as you go towards the edge of the object, like a real dielectric material. Figure 18b starts to look like a refractive object, although we are still missing the light bending. Note that this is usually enough when we model thin objects as zero thickness geometry (like a plane for a window). This obviously does not work for our example of a glass ball.



(a) Transparent ball (b) Transparent ball with reflection



(c) Refractive ball, bidirectional (d) Refractive ball, unidirectional



(e) Refractive ball, transp. shadows

Figure 18: Refraction types.

- Figure 18c is the "right" result with IOR-based reflection and refraction, computed using Walter et al.'s model [Wal+07] [HD14]. Unfortunately, unless under a very specific lighting setup, this is very difficult to render. The render was done using a UPS/VCM integrator.
- Figure 18d is the same render using only unidirectional path tracing. Note the additional noise in the caustics. In this special case, where the geometry is simple and the light big enough, we could render this scene without a bidirectional integrator.
- In production we usually use a mix of the last two methods. Real reflection and refractions for camera paths, but "transparent" (straight, without bending due to changes in IOR) paths for shadowing

(Figure 18e). For this to work correctly, we need to carefully let the bidirectional integrator and lights to work together.

7.3 Thin shadows

If we are not careful, by enabling transparent shadows[2], some light paths will be counted multiple times. The modification needed to make this work turns out to be pretty simple: for a path that is ending on a bidirectional light, we need to disable direct lighting seen through refraction (Figure 19). The trick is to do it if and only if we are doing direct lighting, in order to still see the scene through refraction, for all indirect effects.

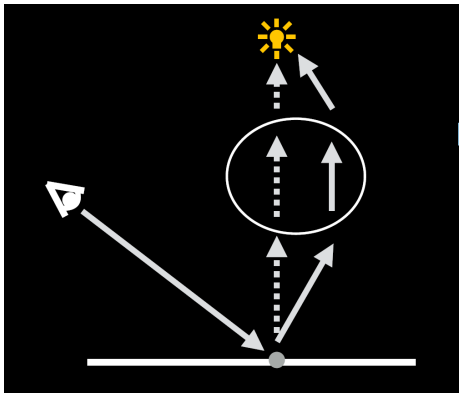


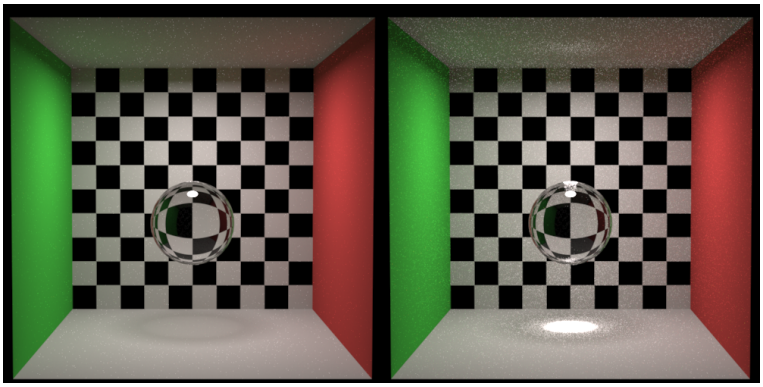
Figure 19: Counting the same path twice.

If we don't do that, the more complex the object is, the more additional light paths we are going to count. It can be bad even for a glass ball which cannot have more than two consecutive refraction events, but as soon as the object is more complicated, like our teapot, it becomes obvious that something is very wrong (Figure 20).
Disabling direct lighting seen through refraction has an unfortunate side effect: even lights seen directly though refraction will disappear. This is because we implicitly assumed that we have transparent shadows, but in order for this assumption to be true, we need at least one reflection event between the camera and the refraction event. Fortunately, by using LPE, we can detect such cases and disable the trick for those paths (Figure 21).

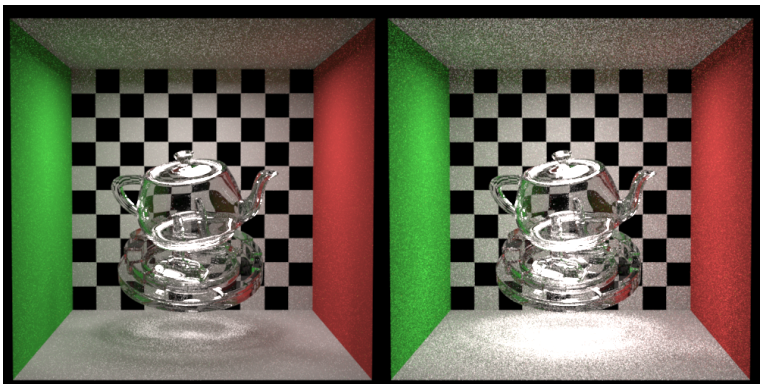
7.4 RenderMan implementation

Bidirectional lights and thin shadows are controlled in RenderMan using a combination of two flags on the lights (`traceLightPaths` and `thinShadows`), and the integrator type. When using a unidirectional integrator like `PxrPathTracer`, the `traceLightPaths` flag on the light is ignored (Figure 22a, Figure 22c). When using a bidirectional integrator like `PxrVCM`, the `traceLightPaths` flag on the light is used to decide if we need to trace light paths from this light (Figure 22b, Figure 22d). As we saw earlier, it is not mandatory to have this flag to see caustics. For example, depending on the lighting setup, a unidirectional render is capable of rendering caustics. With this flag, we let the user decide, per light, where to use the photon or light path budget, and not waste it on lights that don't need it.
The second flag `thinShadows` is only active when `traceLightPaths` is off. For cases where we can't have clean caustics and can't use the bidirectional integrator, turning on this flag will let the integrator compute transparent shadows for this light (Figure 22e, Figure 22f). It will also automatically turn off direct lighting in refraction, and internally use LPEs to track paths that are refraction only.

[2]Called thin shadows in RenderMan, as they mimic the behavior of shadowing through thin refractive objects.

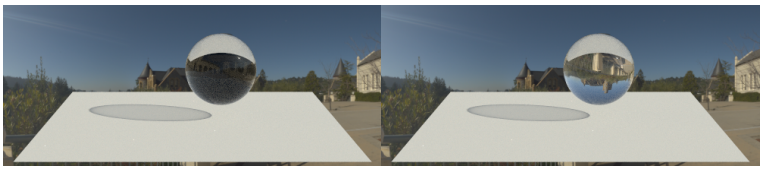


(a) `thinShadows` (b) Double-counted `thinShadows`



(c) `thinShadows` (d) Double-counted `thinShadows`

Figure 20: Thin shadows with correct energy (a & c), incorrectly double-counted energy (b & d).

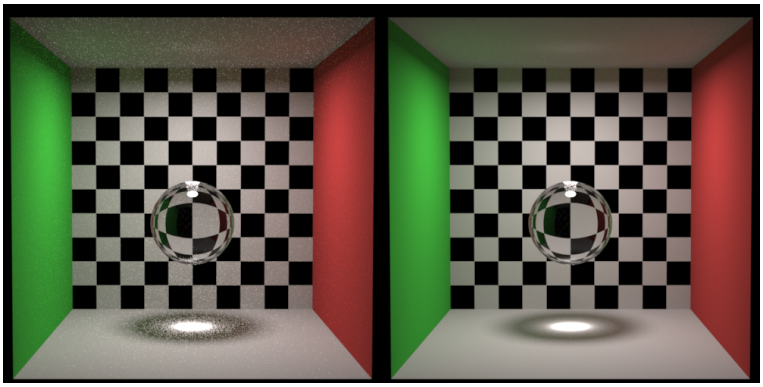


(a) (b)

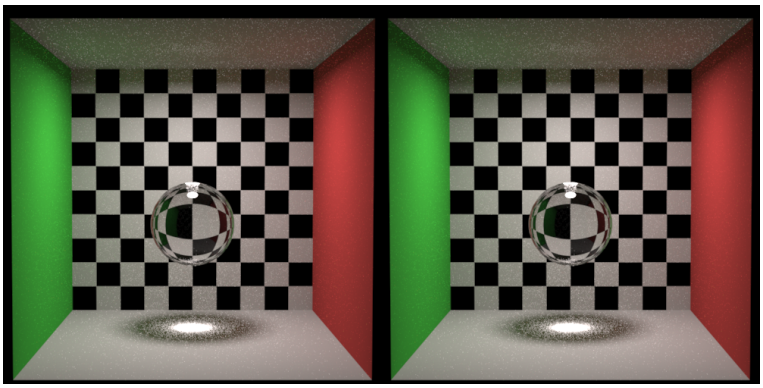
Figure 21: Disabling `thinShadows` for path C<TS>*L.

8 Conclusion

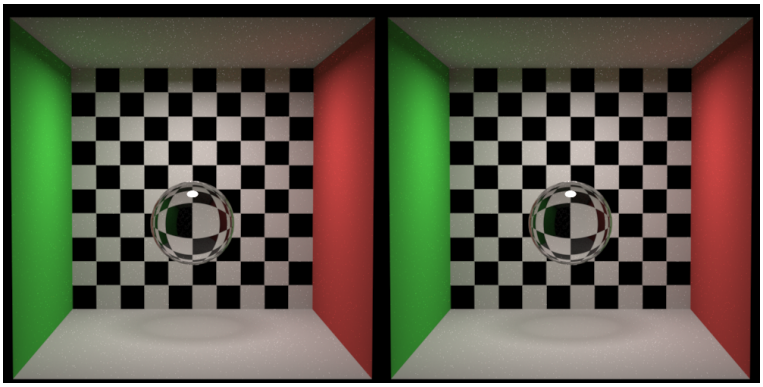
Rendering is now a sampling problem, as shading and lighting are physically based. But this does not mean that everything must become photorealistic and absolutely physically correct. The goal is still to render visually pleasing images, so being physically based should not prevent artistic creativity.
There are just a few rules we need to keep in mind when bending reality. If there is no way to correctly sample a hack, it might not be a good one. When adding a new feature, we should always keep in mind what the function we are evaluating is, and decouple it from how we are computing it.
We showed that we can still cheat to render faster, enable more controllable refractions, and that LPEs are a powerful tool to describe what we are trying to modify. By working on light paths, in a way that is decoupled from the integration method, these techniques are compatible with unidirectional as well as bidirectional renders.



(a) Unidir., `traceLightPaths` on (b) Bidir., `traceLightPaths` on



(c) Unidir., `traceLightPaths` off (d) Bidir., `traceLightPaths` off



(e) Unidir., `thinShadows` on (f) Bidir., `thinShadows` on

Figure 22: RenderMan `thinShadows` and `traceLightPaths` flags.

9 Acknowledgments

We would like to thank the RUKUS team, *Finding Dory*'s production, and in particular its lighting and shading artists, for their willingness to adopt a brand new back-end pipeline, as well as the RenderMan group, for making it possible in the first place to render of all of these new ray-traced images.

References

[Geo+12] I. Georgiev, J. Křivánek, T. Davidovič, and P. Slusallek. "Light Transport Simulation with Vertex Connection and Merging". In: *ACM Trans. Graph.* 31.6 (Nov. 2012), 192:1–192:10. ISSN: 0730-0301. DOI: 10.1145/2366145.2366211. URL: http://doi.acm.org/10.1145/2366145.2366211.

[Gri] L. Gritz. *OSL (Open Shading Language)*. URL: https://github.com/imageworks/OpenShadingLanguage/wiki/OSL-Light-Path-Expressions.

[HD14] E. Heitz and E. D'Eon. "Importance Sampling Microfacet-Based BSDFs using the Distribution of Visible Normals". In: *Computer Graphics Forum* 33.4 (July 2014), pp. 103–112. DOI: 10.1111/cgf.12417. URL: https://hal.inria.fr/hal-00996995.

[Hec90] P. S. Heckbert. "Adaptive radiosity textures for bidirectional ray tracing". In: *SIGGRAPH Comput. Graph.* 24.4 (Sept. 1990), pp. 145–154. ISSN: 0097-8930. DOI: 10.1145/97880.97895. URL: http://doi.acm.org/10.1145/97880.97895.

[HPJ12] T. Hachisuka, J. Pantaleoni, and H. W. Jensen. "A Path Space Extension for Robust Light Transport Simulation". In: *ACM Trans. Graph.* 31.6 (Nov. 2012), 191:1–191:10. ISSN: 0730-0301. DOI: 10.1145/2366145.2366210. URL: http://doi.acm.org/10.1145/2366145.2366210.

[HV13] C. Hery and R. Villemin. "Physically Based Lighting at Pixar". In: *Physically Based Shading in Theory and Practice, ACM SIGGRAPH 2013 Courses*. Anaheim, CA, USA, 2013. URL: http://graphics.pixar.com/library/PhysicallyBasedLighting/index.html.

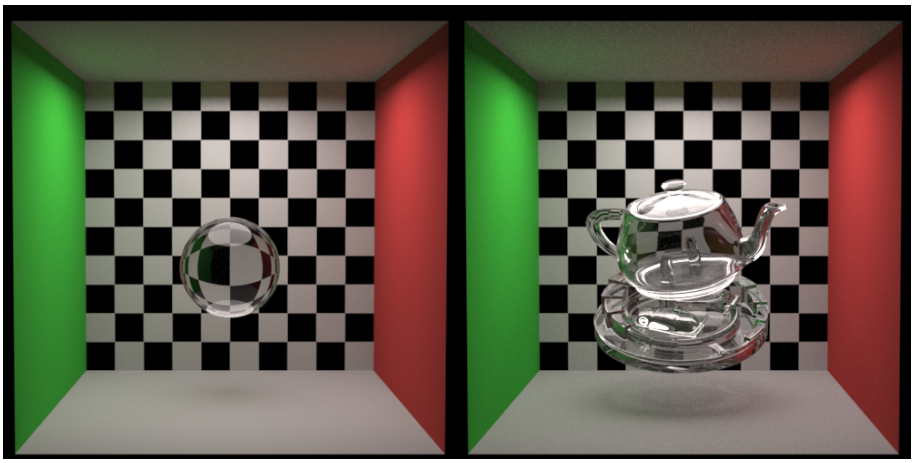
[Kaj86] J. T. Kajiya. "The Rendering Equation". In: *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), pp. 143–150. ISSN: 0097-8930. DOI: 10.1145/15886.15902. URL: http://doi.acm.org/10.1145/15886.15902.

[PH14] L. Pekelis and C. Hery. *A statistical framework for comparing importance sampling methods, and an application to rectangular lights*. Emeryville, CA, USA, 2014. URL: http://graphics.pixar.com/library/StatFrameworkForImportance/paper.pdf.

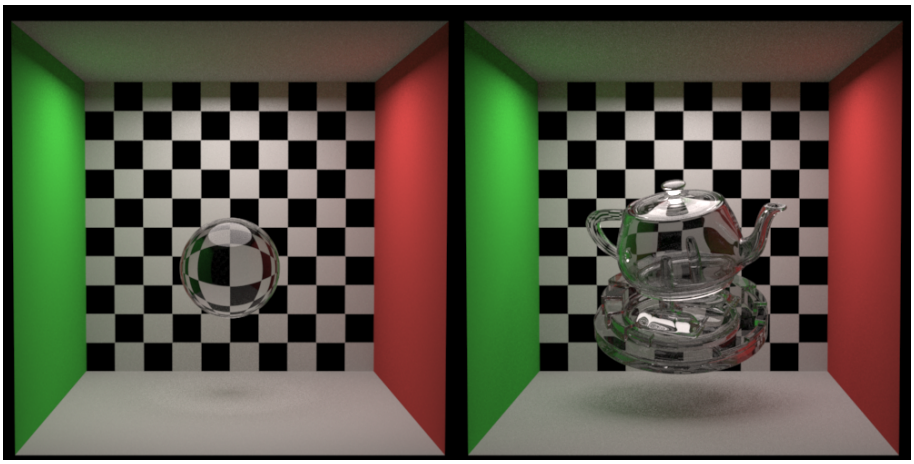
[VG95] E. Veach and L. J. Guibas. "Optimally Combining Sampling Techniques for Monte Carlo Rendering". In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: ACM, 1995, pp. 419–428. ISBN: 0-89791-701-4. DOI: 10.1145/218380.218498. URL: http://doi.acm.org/10.1145/218380.218498.

[VH15] R. Villemin and C. Hery. "RenderMan RIS". In: *Art and Technology at Pixar, from Toy Story to Today, ACM SIGGRAPH Asia 2015 Courses*. SA '15. Kobe, Japan: ACM, 2015, 5:1–5:89. ISBN: 978-1-4503-3924-7. DOI: 10.1145/2818143.2818155. URL: http://doi.acm.org/10.1145/2818143.2818155.

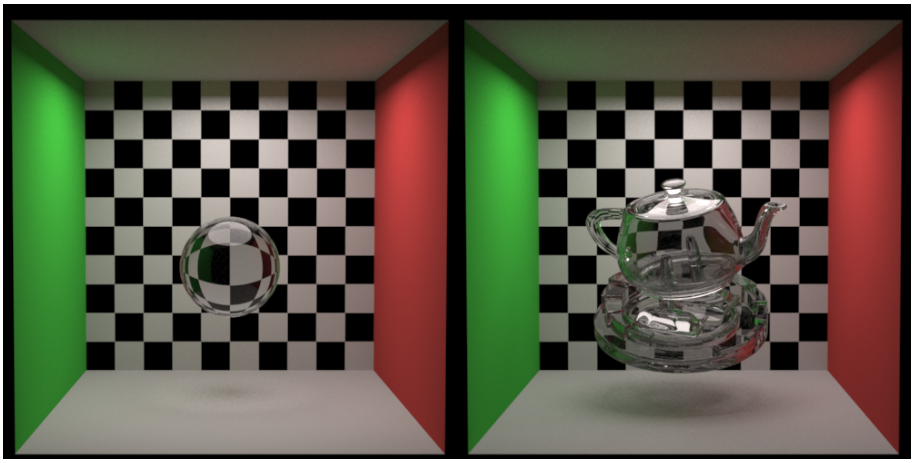
[Wal+07] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. "Microfacet Models for Refraction Through Rough Surfaces". In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. Grenoble, France: Eurographics Association, 2007, pp. 195–206. ISBN: 978-3-905673-52-4. DOI: 10.2312/EGWR/EGSR07/195-206. URL: http://dx.doi.org/10.2312/EGWR/EGSR07/195-206.



(a) Glass ball, `thinShadows` (b) Glass teapot, `thinShadows`

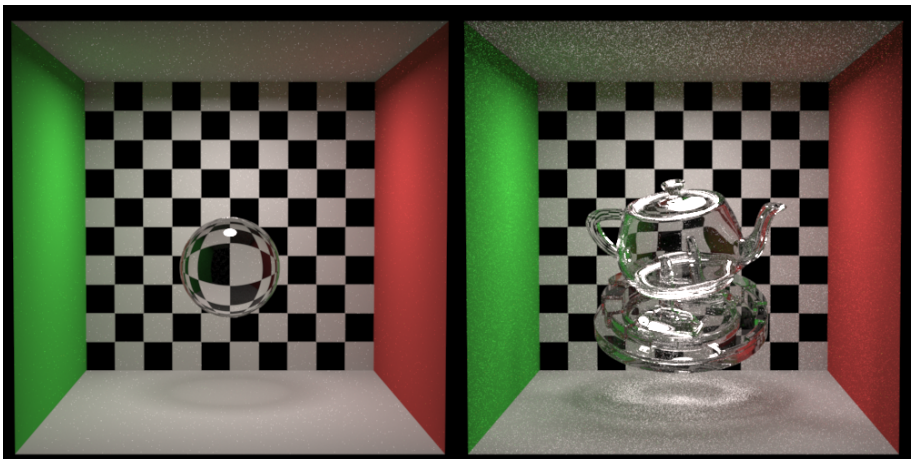


(c) Glass ball, unidirectional (d) Glass teapot, unidirectional

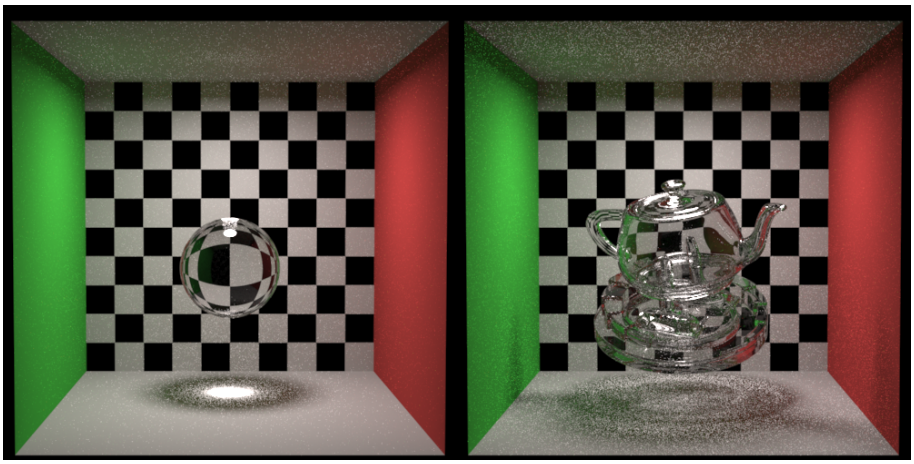


(e) Glass ball, bidirectional (f) Glass teapot, bidirectional

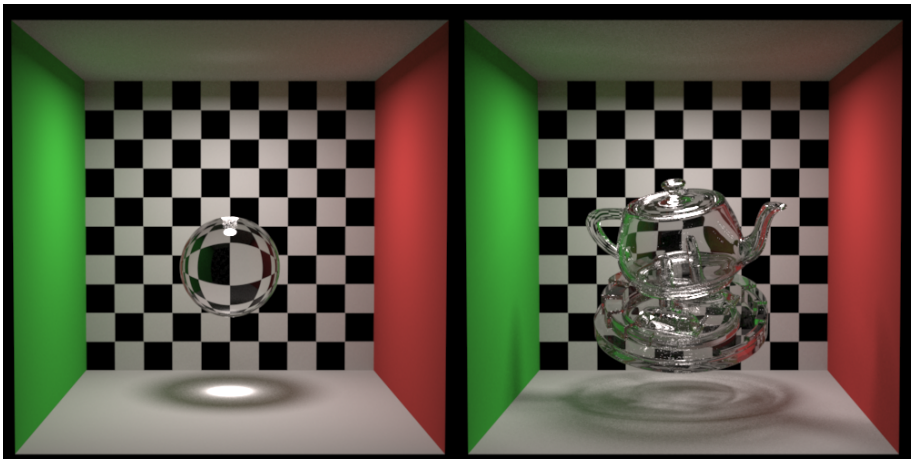
Figure 23: Large light source.



(a) Glass ball, `thinShadows` (b) Glass teapot, `thinShadows`

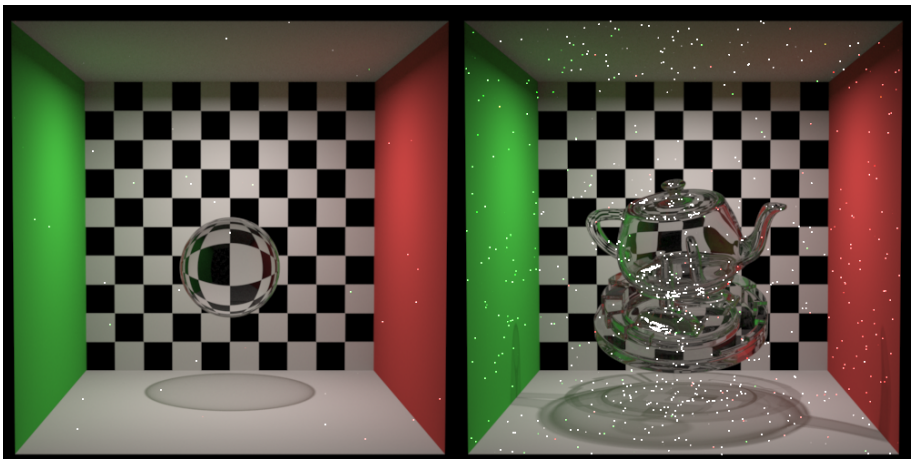


(c) Glass ball, unidirectional (d) Glass teapot, unidirectional

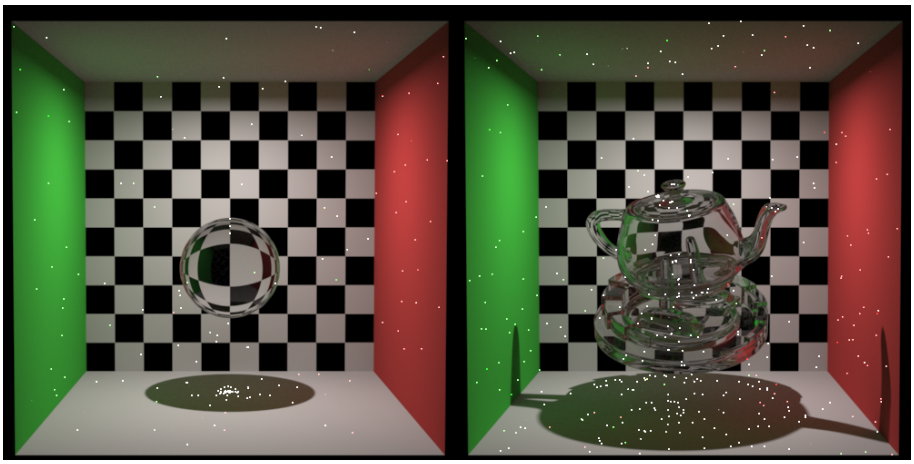


(e) Glass ball, bidirectional (f) Glass teapot, bidirectional

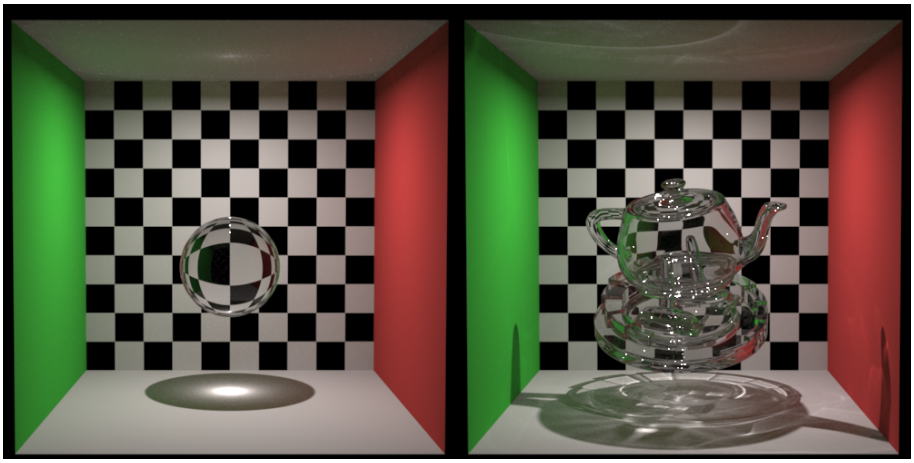
Figure 24: Medium light source.



(a) Glass ball, `thinShadows` (b) Glass teapot, `thinShadows`



(c) Glass ball, unidirectional (d) Glass teapot, unidirectional



(e) Glass ball, bidirectional (f) Glass teapot, bidirectional

Figure 25: Small light source.