CHAPTER 17

Ignoring the Inconvenient When Tracing Rays

Matt Pharr
NVIDIA

ABSTRACT

Ray tracing's greatest strength—that it can simulate all types of light transport—can also be its greatest weakness: when there are a few paths that unexpectedly carry much more light than others, the produced images contain a smattering of pixels that have bright spiky noise. Not only can it require a prohibitive number of additional rays to average out those spikes, but those pixels present a challenge for denoising algorithms. This chapter presents two techniques to address this problem, preventing it from occurring in the first place.

17.1 INTRODUCTION

Ray tracing is a marvelous algorithm, allowing unparalleled fidelity in the accurate simulation of light transport for image synthesis. No longer are rasterizer hacks required to generate high-quality images; real-time graphics programmers can now happily move forward to a new world, tracing rays to make beautiful images, and be free of the shackles of that history.

Now, let us move on to the new hacks.

17.2 MOTIVATION

Figure 17-1 presents two images of a pair of spheres in a box, both rendered with path tracing, using a few thousand paths per pixel to compute a high-quality reference image. The scene is illuminated by an area light source (not directly visible). The only difference between the two images is the material on the right-hand sphere: diffuse on the left and a perfectly specular mirror on the right.

E. Haines, T. Akenine-Möller (eds.), *Ray Tracing Gems*, https://doi.org/10.1007/978-1-4842-4427-2_17

Something interesting happens if we render these scenes with a more realistic number of samples per pixel—here we used 16.[1] Figure 17-2 shows the result: the scene with only diffuse spheres looks pretty good. However, the scene with the mirrored sphere has bright spiky noise, sometimes called "fireflies," scattered all around the scene.

From what one might have thought is an innocuous change in material, we see a massive degradation in image quality. What is going on here?

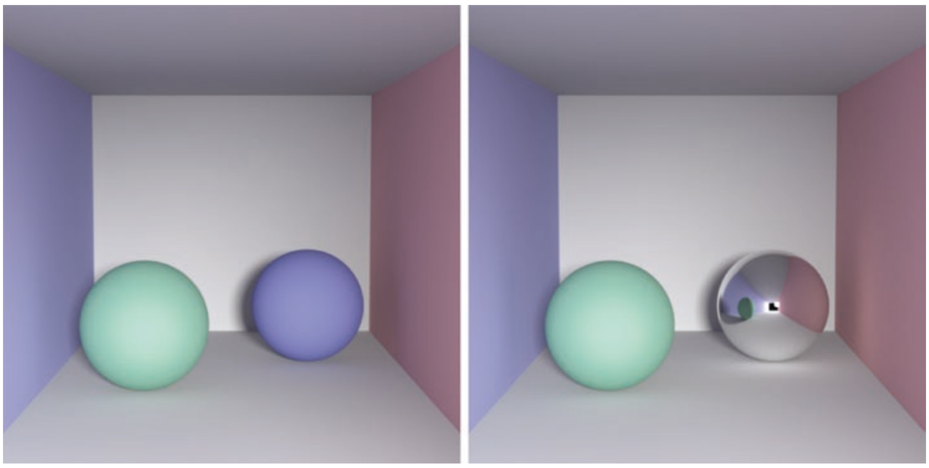

Figure 17-1. *A simple scene, illuminated by a single area light source, rendered with path tracing and enough paths to give high-quality reference images. The only difference between the two renderings is that one of the diffuse spheres has been changed to be mirrored in the right image.*

To understand what has happened, imagine for a moment that we instead wanted to compute the average pixel value of an image by averaging pixel values at just a handful of randomly chosen pixels. Consider performing that task with the two converged images in Figure 17-1. In the scene with two diffuse spheres, most pixels have values that are roughly the same magnitude. Thus, no matter which pixels you choose, the average you compute will be in the correct ballpark.

In the scene with the specular sphere, note that we can see a small reflection of the light source in the mirrored sphere. In the rare cases where we happened to choose for our computation one of the pixels where the light source is visible, we would be adding in the amount of that light's emission; most of the time, however, we would miss it entirely.

[1]While 16 samples per pixel may be impractical today for interactive graphics for complex scenes at high resolutions, we assume that both temporal accumulation of samples and a denoising algorithm will be used in practice. In this chapter, we will not use either in order to make it easy to understand the image artifacts.

Given the small size of the light and its distance from the scene, the light source needs a fairly large amount of emission to give enough illumination to light the scene. Here, in order to have final shaded pixel values roughly in the range [0, 1], the light's emission has to be (500,500,500) in RGB. Thus, if we happen to include a pixel where the light's reflection is visible but sample only a small number of pixels, we will grossly overestimate the true average. Most of the time, when we do not include one of those pixels, we will underestimate the average, since we are not including any of the pixels with high values.

Now back to the rendered images in Figure 17-2. When path tracing, at each point on a surface where we trace a new ray, we face more or less the same problem as in the image averaging exercise: we are trying to estimate a cosine and BSDF-weighted average of the light arriving at the point using just a few rays. When the world is mostly similar in all directions, choosing just one direction works well. When it is quite different in a small set of directions, we run into trouble, randomly getting much too high estimates of the average for a small fraction of the pixels. In turn, that manifests itself as the kind of spiky noise we see on the right in Figure 17-2.

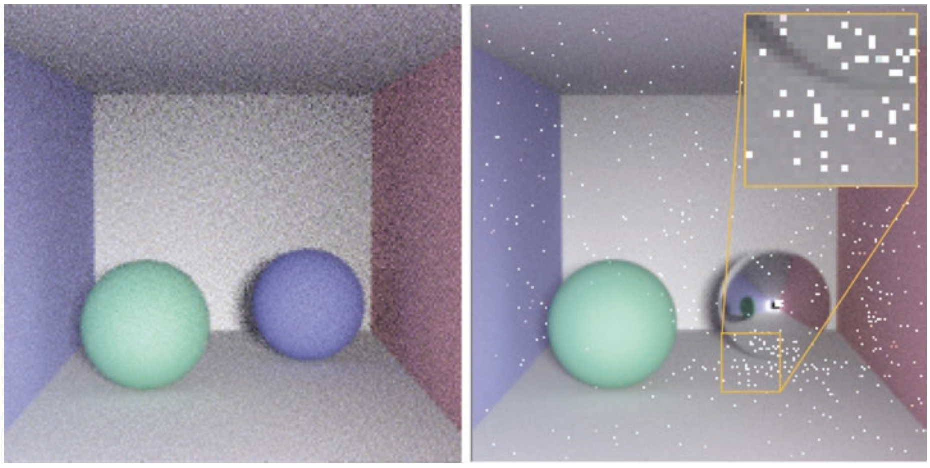

Figure 17-2. *Example scenes, rendered with 16 samples per pixel. Left: the scene with diffuse spheres is well-behaved and could easily be denoised to a high-quality image (of a boring scene). Right: we have a multitude of spiky noisy pixels and some way to go before we have a good-looking image.*

Understanding the cause of the spiky noise, we can see something interesting in the distribution of the speckles: they are much more common on surfaces that can "see" the mirrored sphere, as a path has to hit the mirrored sphere in order to unexpectedly find its way back to the light source. Note that there is a kind of shadow of no speckles in the lower left of the image; the green sphere occludes points there from seeing the mirrored sphere directly.

The challenging thing about this kind of noise is how slowly it goes away as you take more samples. Consider the case of computing the average image color again: once we include one of those (500,500,500) colors in our sum, it takes quite a few additional samples in the range [0, 1] to get back to the true average. As it turns out, taking more samples can make the image look *worse*, even though it is (on average) getting better: as more rays are traced, more and more pixels will have paths that randomly hit the light.

17.3 CLAMPING

The simplest solution to this problem is *clamping*. Specifically, we clamp any sample values c that are higher than a user-provided threshold t. Here is the algorithm in its entirety:

$$c' = \min(c, t). \tag{1}$$

Figure 17-3 shows the mirrored sphere scene rendered with path contributions clamped at 3. Needless to say, it is much less noisy. The image on the left was rendered with 16 samples per pixel (like the images in Figure 17-2) and one on the right was rendered to convergence.

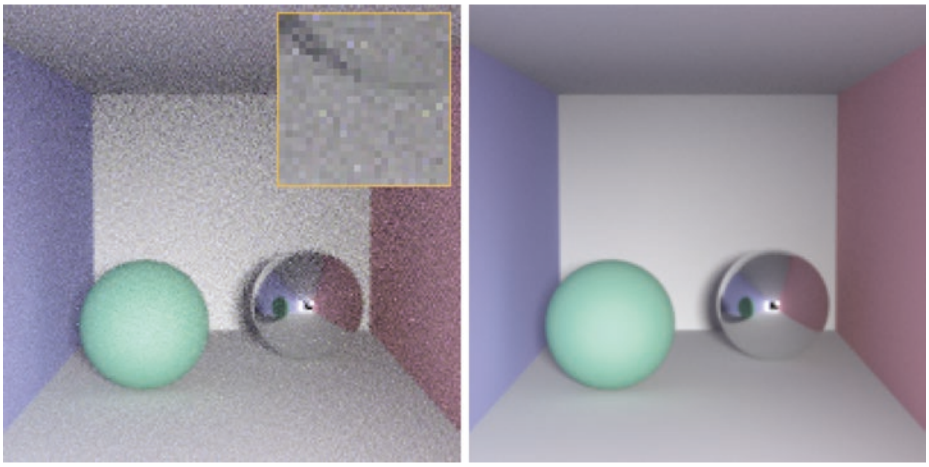

Figure 17-3. *Specular sphere scene rendered with clamping using 16 (left) and 1024 (right) samples per pixel. Note that the spiky noise from Figure 17-2 has disappeared, though we have also lost the light reflected by the sphere onto the floor and wall next to it (visible in Figure 17-1).*

With 16 samples, the spiky noisy pixels are gone, and we are much closer to a good-looking image. However, note the difference between the 1024-sample image here and the final image in Figure 17-1: we have lost the focused light from the light source below the mirrored sphere on the floor and to the right on the wall (a so-called *caustic*). What is happening is that the illumination comes from a small number of high-contribution paths and, thus, clamping prevents them from contributing much to the final image.

17.4 PATH REGULARIZATION

Path regularization offers a less blunt hammer than clamping. It requires slightly more work to implement, but it does not suffer from the loss of energy that we saw with clamping.

Consider again the thought exercise of computing the average value of the image from just a few pixels: if you have an image with a few very bright pixels, like we have with the reflection of the light source in the mirrored sphere, then you could imagine that you would get a better result if you were able to apply a wide blur to the image before picking pixels to average. In that way, the bright pixels are both spread out and made dimmer, and thus the blurred image has less variation and which pixels you choose matters less.

Path regularization is based on this idea. The concept is straightforward: blur the BSDFs in the scene when they are encountered by indirect rays. When regularization is performed at such points, the sphere becomes glossy specular rather than perfectly specular.

The left image in Figure 17-4 shows how this works with our scene at 16 samples per pixel, and the right one shows its appearance as the image converges at around 128 samples. Regularization has eliminated the spiky noise while still preserving a representation of the caustic reflection of the light source.

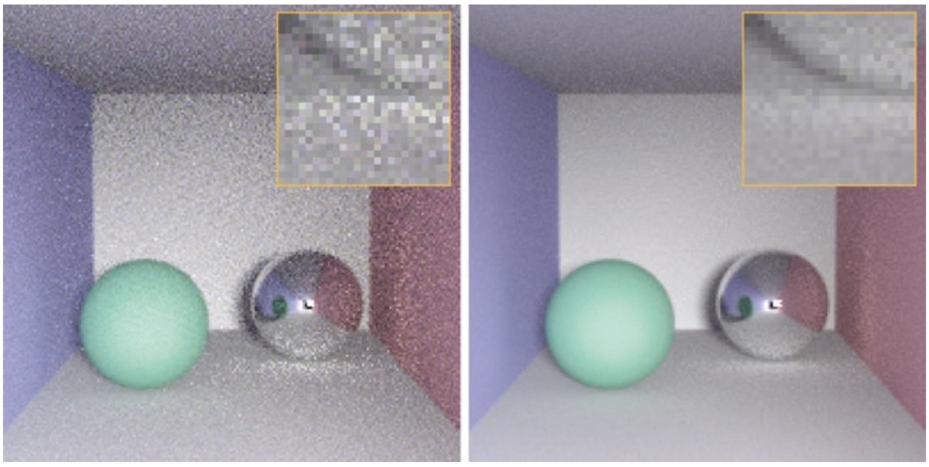

Figure 17-4. *Left: the scene is rendered with path regularization at 16 samples per pixel. Note that the random spiky noise is gone, while the caustic from the light source is still present. Right: once we accumulate 128 samples per pixel, we have a fairly clean image that still includes the caustic.*

17.5 CONCLUSION

Sometimes in ray tracing we encounter spiky noise in our images due to localized bright objects or reflections that are not being well-sampled by the employed sampling techniques. Ideally, we would improve our sampling techniques in that case, but this is not always possible or there is not always time to get it right.

In those cases, both clamping and path regularization can be effective techniques to get good images out the door; both are easy to implement and both work well. Clamping is a one-line addition to a renderer, and path regularization just requires recording whether a non-specular surface has been encountered in a ray path and then, when so, making subsequent BSDFs less specular.

The path regularization approach can be placed on a much more principled theoretical ground than we have used in describing it here. See Kaplanyan and Dachsbacher's paper [1] for details.

A more principled approach to clamping is *outlier rejection*, where samples that are unusually bright with respect to other samples are discarded. Outlier rejection is more robust than a fixed clamping threshold and loses less energy. See the paper by Zirr et al. [2] for a recent outlier rejection technique that is amenable to GPU implementation.

REFERENCES

[1] Kaplanyan, A. S., and Dachsbacher, C. Path Space Regularization for Holistic and Robust Light Transport. *Computer Graphics Forum 32*, 2 (2013), 63–72.

[2] Zirr, T., Hanika, J., and Dachsbacher, C. Reweighting Firefly Samples for Improved Finite-Sample Monte Carlo Estimates. *Computer Graphics Forum 37*, 6 (2018), 410–421.