Foundations and Trends® in
Computer Graphics and Vision
Vol. 10, No. 2 (2014) 103–175

DOI: 10.1561/0600000073

now
the essence of knowledge

The Path to Path-Traced Movies

Per H. Christensen
Pixar Animation Studios
per@pixar.com

Wojciech Jarosz
Dartmouth College
wojciech.k.jarosz@dartmouth.edu

Contents

Abstract

Path tracing is one of several techniques to render photorealistic images by simulating the physics of light propagation within a scene. The roots of path tracing are outside of computer graphics, in the Monte Carlo simulations developed for neutron transport. A great strength of path tracing is that it is conceptually, mathematically, and often-times algorithmically simple and elegant, yet it is very general. Until recently, however, brute-force path tracing techniques were simply too noisy and slow to be practical for movie production rendering. They therefore received little usage outside of academia, except perhaps to generate an occasional reference image to validate the correctness of other (faster but less general) rendering algorithms. The last ten years have seen a dramatic shift in this balance, and path tracing techniques are now widely used. This shift was partially fueled by steadily increasing computational power and memory, but also by significant improvements in sampling, rendering, and denoising techniques. In this survey, we provide an overview of path tracing and highlight important milestones in its development that have led to it becoming the preferred movie rendering technique today.

P. H. Christensen and W. Jarosz. *The Path to Path-Traced Movies*. Foundations and Trends® in Computer Graphics and Vision, vol. 10, no. 2, pp. 103–175, 2014. DOI: 10.1561/0600000073.

1

Introduction

Rendering computer-generated (CG) movies is tough. There are 130,000 high-resolution frames in a 90-minute movie, with each frame requiring computation of typically two million pixel colors (many more for IMAX movies). This is several hundred billion pixel colors that will be scrutinized by the movie director and by audiences worldwide. The images are often computed with motion blur and depth of field, to mimic these characteristic effects (limitations) of real cameras. The images must be free of visual noise—one particularly pesky type of noise is occasional bright pixels known as "fireflies". There can be no spatial or temporal aliasing (affectionately known as "jaggies" and "crawlies") in the images. The color of each pixel depends not only on what object is visible in that pixel (including its orientation, material, texture, illumination, etc.), but also—through shadows and reflected light—on objects in other parts of the scene. The surface color calculations have to be programmable, with the computations specified in standalone programs called "shaders". Typical scenes contain huge amounts of geometry and texture data, often straining the available memory even on high-end computers. There are often dozens of textures specifying the material properties of each surface, and a scene can contain

Figure 1.1: Frames from the movies *Toy Story* (1995) and *Finding Dory* (2016). (© 1995, 2016 Pixar/Disney.)

millions of surfaces. With all the data that goes into rendering each frame, production-strength renderers are sometimes jokingly referred to as data management systems with images as a by-product. Rendering times are crucial as well, both for quick test images during the development of the movie, and for the final-quality frames that will appear in movie theaters.

Figure 1.1(a) shows a frame from the first computer-generated animated feature film: Pixar's *Toy Story* from 1995. This movie was rendered with the RenderMan renderer using the Reyes scan-line algorithm [25] with shadow maps and reflection maps [4]. For many years, the Reyes algorithm was the work-horse of most CG and visual effects work at major studios.

The last 15 years has seen hybrid renderers combining the Reyes algorithm for directly visible objects with ray-traced shadows and reflections. Soft indirect illumination has been computed with a variety of methods, including distribution ray tracing and point-based global illumination.

At the same time, other renderers, such as Arnold, pushed for a complete switch to path tracing. Compared to Reyes-based hybrid renderers, path tracing is a simpler and more brute-force approach. It has its roots in a statistical sampling method called Monte Carlo, which was first used for particle simulation in nuclear engineering. Path tracing can render shadows and reflections in a conceptually simple recursive

manner, but on the other hand it is more noisy and less memory efficient than some Reyes hybrids. Path tracing is not necessarily the fastest method to render final movie-quality images with indirect illumination, depth of field, and motion blur: for example, point-based global illumination has no noise, and distribution ray tracing with irradiance gradients and radiosity caching is better able to exploit sample domain coherency. However, path tracing's natural ability to handle complex light transport effects, along with its potential to simplify the production pipeline, reduce iteration time during layout and lighting, and improve overall workflow, are enticing advantages.

Figure 1.1(b) shows a frame from the recent movie *Finding Dory*. Here all the direct visibility, shadows, reflections, refractions, indirect diffuse illumination, and subsurface scattering are computed with RenderMan's implementation of path tracing.

Even though the algorithmic developments for the switch to path tracing have been under way for quite some time, there is a fairly sudden wave of studios switching their pipelines over. One might even talk about a path tracing "revolution" [61]. This article is our attempt to retrace the steps the industry has taken on its journey to path-traced movies. We will identify major hurdles that stood in the way of that transition, describe the technical milestones that pushed the field forward over the last couple of decades, and discuss the combination of circumstances that came together to propel the CG and VFX movie industry into a path-traced world. Since the journey is not yet complete, we will also discuss on-going challenges and open questions that practitioners and researchers will need to address in the years to come.

2

Illumination

Rendering programs typically divide the light transport into illumination that comes directly from the light sources and is reflected exactly once before reaching the camera, and indirect illumination that has been reflected multiple times.

2.1 Direct and indirect illumination

Figure 2.1 shows four images of a diffuse box with chrome and glass teapots illuminated by a light source in the ceiling. Figure 2.1(a) shows only the bright square light source in the ceiling. Figure 2.1(b) shows direct illumination: diffuse reflection by the walls and floor, and specular reflection (small white highlights) by the chrome and glass teapots. Figure 2.1(c) shows indirect illumination in the same scene. The most visible effects are: the ceiling is now illuminated (mostly by light bouncing off of the walls and floor), the white surfaces are tinted red and blue, specular reflection in the chrome teapot, specular refraction through the glass teapot, and a bright "puddle" of focused light under the glass teapot. Figure 2.1(d) shows all illumination in the scene: the light source, direct and indirect illumination from images (a)–(c).

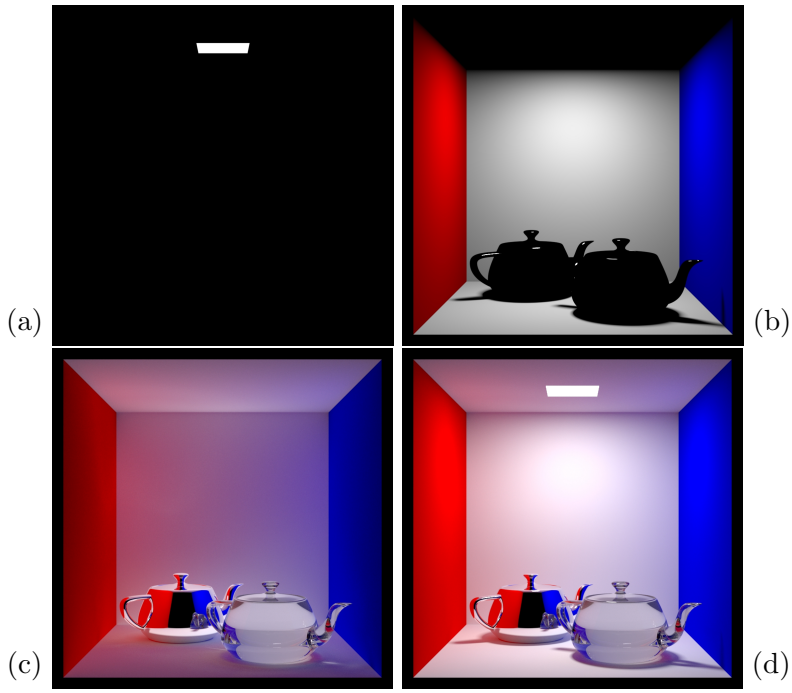

Figure 2.1: Illumination components: (a) Light source. (b) Direct illumination. (c) Indirect illumination. (d) All illumination.

2.2 Indirect illumination types

The three images in Figure 2.2 show different components of indirect illumination.

A simple real-life example of indirect diffuse (also known as "global illumination", "soft indirect", "diffuse interreflection", or "color bleeding") is a white carpet next to a matte red wall. Light being reflected by the wall is red, and gives the white carpet a pink hue. This effect is shown in Figure 2.2(a) where the red and blue colors from the walls bleed onto the white ceiling, back wall, and floor. For a long time, indirect diffuse was faked in movie rendering by manually adding direct illumination from additional colored lights (or it was ignored).

Figure 2.2: Indirect illumination components: (a) Indirect diffuse. (b) Specular reflections and refractions of diffusely reflected light. (c) Diffuse reflection of specular reflections and refractions (caustics).

Specular reflection or refraction of diffusely reflected light shows the diffuse surfaces reflected in smooth surfaces or refracted through glass, water, etc. See Figure 2.2(b) for an example.

Light that travels from the light source through one or more specular reflections or refractions and then hits a diffuse surface are called "caustics". Common real-life examples are the bright spot on a tablecloth next to a candle-lit glass, or inside a metal ring resting on a rough surface. Caustics are shown in Figure 2.2(c): the most prominent is the previously mentioned focused light under the glass teapot, but also note the bright region around the chrome teapot, and the subtle ring of light from the chrome teapot lid onto the ceiling and back wall.

The textbooks by Cohen and Wallace [21], Glassner [35], and Dutré et al. [30] provide good overviews of computation methods for indirect illumination.

3

Path Tracing

This chapter describes the origins of path tracing, and gives a simple overview of path tracing and how it has been used in movies.

3.1 Origins of path tracing

The very first Monte Carlo simulations were done by Enrico Fermi using a small mechanical adding machine in the 1930s [84], but Monte Carlo only became more widely used as the first electronic computers were developed. Monte Carlo simulations were especially useful for neutron transport simulation in the development of nuclear bombs and reactors in the 1940s. Much inspiration can be found in classic texts such as Spanier and Gelbard [114], and Kalos and Whitlock [59].

Photons are governed by the same equations as neutrons (they are both electrically neutral—each particle neither attracts nor repels others of the same kind), so the same mathematical simulation methods can be used. Path tracing is a variation of Monte Carlo simulation that was introduced to computer graphics by Kajiya [57] in 1986, and the connection to particle transport was later solidified by Arvo and Kirk [5].

3.2 Simple path tracing

Path tracing is a simple recursive method. To calculate the color of an image pixel, a number of rays are traced from the eye point through that pixel. When a ray intersects a surface, the direct illumination from the light sources is calculated for the intersection point (which includes tracing shadow rays between the light sources and intersection point—referred to as "next-event estimation" in the particle transport field). In addition, a new ray is spawned to calculate indirect illumination. The direction of the new ray is stochastically chosen based on the light scattering properties of the surface material: specular or matte, reflective or refractive. When such a ray hits another surface the direct illumination is calculated there (including tracing more shadow rays), a new ray are spawned, and so on. The colors from all rays at all depths contribute to the color of the pixel of their originating eye ray. The recursion stops when the new ray does not hit a surface, or it is probabilistically terminated using Russian roulette [5], or when a pre-determined maximum recursion depth has been reached.

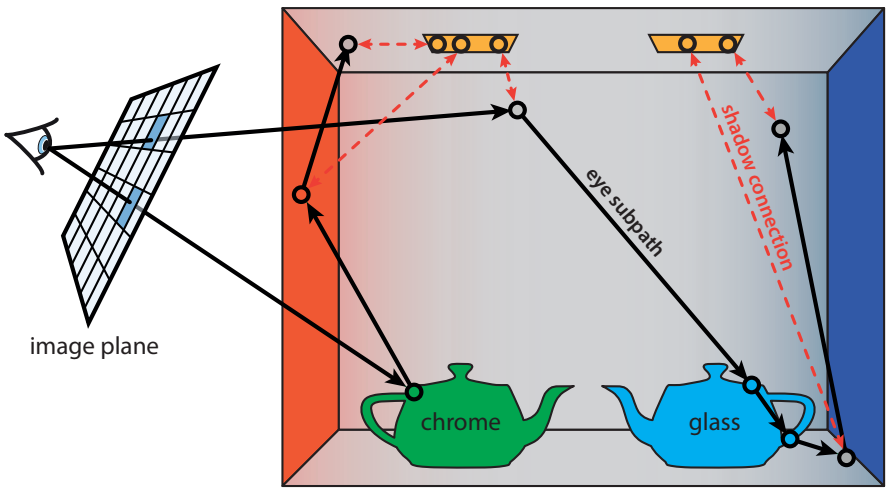

Figure 3.1: An illustration of tracing paths from the eye to the light sources in a Cornell box scene with two teapots.

Figure 3.1 illustrates this process in a variation of the classic Cornell box with two teapots and two light sources. One path originating from the eye hits the chrome teapot. Chrome exhibits specular reflection so a reflection ray is generated (in the mirror direction), which subsequently hits the diffuse red wall. At that point a random position on one of the light sources is chosen, a shadow ray (shown in red) is shot to determine visibility, and the direct illumination from that light source point is calculated. The path continues as a new random diffuse reflection ray is chosen, which this time hits the ceiling. A new shadow ray is shot, and the path is stochastically terminated. The other path from the eye first hits the diffuse back wall, direct illumination is calculated using a shadow ray to determine visibility, and a new reflection direction is chosen. That ray hits the glass teapot, where a choice is made between specular reflection or refraction, and refraction is chosen. After one more refraction the path hits the diffuse floor, a shadow ray is traced, and a new ray direction is stochastically chosen. The new ray hits the back wall, another shadow ray is traced, and this path is terminated.

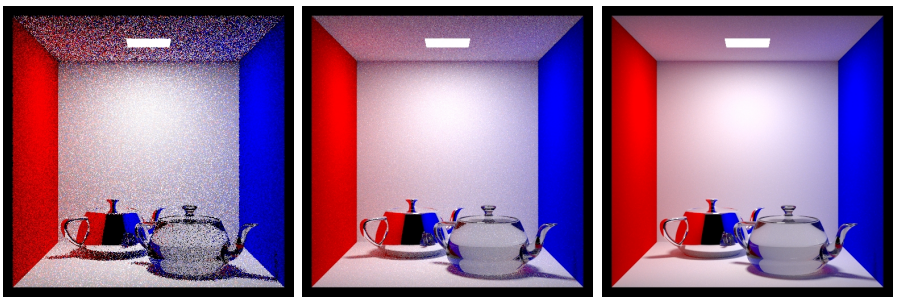

Figure 3.2: Path-traced images rendered with 1, 16, and 256 samples per pixel.

Figure 3.2 shows path-traced images with 1, 16, and 256 samples per pixel in the box scene from figures 2.1 and 2.2. The image on the left was rendered very quickly, but is very noisy; the other images took longer to render but are much less noisy. Despite the noise, an artist can form an opinion about the illumination and materials in the scene even with only a few samples per pixel, i.e. after a few iterations in an interactive session.

When shadows, reflections, refractions, and indirect diffuse are computed in a single pass, *all* geometry needs to be accessible during the entirety of rendering. This is very different from Reyes, and puts a harder constraint on scene size (or requires a computer with more memory).

3.3 Depth of field and motion blur

Real cameras have finite aperture openings and the shutter is open for a finite amount of time. Hence real images have limited depth of field and contain motion blur. In fields such as architectural visualization these effects are not important, but movie audiences expect these effects in CG and VFX images. Figure 3.3 shows examples of these effects: in the left image the front teapot is in focus while the back teapot is out of focus; in the right image both teapots are moving and blurry while the checkered plane is static and sharp.

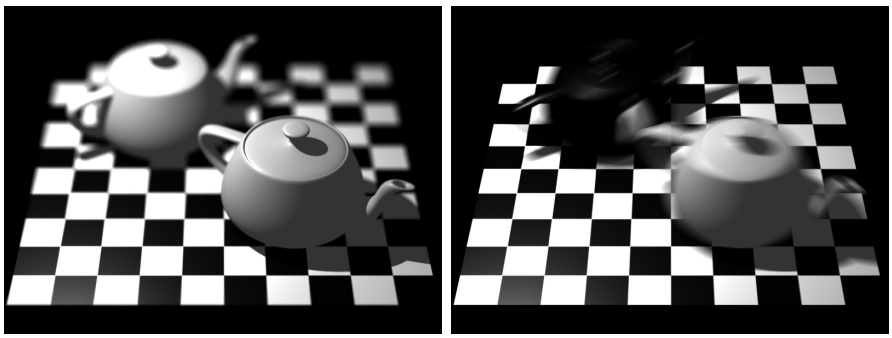

Figure 3.3: (a) Depth of field. (b) Motion blur.

Depth of field can be simulated in path tracing by—instead of shooting eye rays from a single viewpoint—shooting camera rays with small variations in origins and directions [24, 67]. Motion blur can be simulated by assigning each camera ray a random time within the time interval the shutter is open [24] (spawned rays inherit the time of their parent ray). For ray intersection tests, moving or deforming objects have to be transformed to the position and shape corresponding to the ray's time.

3.4 Path tracing in movies

Figure 3.4(a) shows a frame from the short film *Bunny* from Blue Sky Studios, released in 1998. This was an early ground-breaking use of path tracing, proving that effects like indirect diffuse illumination, depth of field, and motion blur can successfully be rendered at film quality using path tracing.

Figure 3.4(b) shows a frame from the movie *Monster House* from 2006. This was the first feature-length movie to be rendered with path tracing; it used the Arnold renderer. The movie was rendered without motion blur—partially as an artistic choice to mimic the look of classic stop-motion films, and partially to avoid noise from sampling the motion. Depth of field (as seen in the image) was added in a separate post-process. For a long time Arnold was the only production-strength renderer based on path tracing, but recently at least a dozen other commercial and in-house renderers have been written or re-written to use path tracing.

Figure 3.4: Path-traced images from *Bunny* and *Monster House*. (Bunny: © 1998 Twentieth Century Fox. All rights reserved. Courtesy of Twentieth Century Fox. Monster House: © 2006 Columbia Pictures Industries, Inc. and GH One LLC. All rights reserved. Courtesy of Columbia Pictures.)

4

Other Rendering Techniques: A Retrospective

While a few companies dove head-first into ray tracing and path tracing, the vast majority of the industry stayed with the Reyes-style rendering approach for many years. For the industry as a whole, the so-called path tracing revolution was more a steady evolution. There are many compounding reasons for the gradual switch, ranging from the memory and computational demands of path tracing in complex production scenes, to the immense engineering and coordination effort of overhauling production pipelines that were entrenched in the Reyes world. Due to these constraints, the industry and research communities were strongly motivated to find alternatives to full-blown path tracing—alternatives which could introduce the benefits of path tracing (e.g. indirect diffuse illumination, soft shadows, recursive reflection and refraction), but as additions incorporated into existing Reyes systems. To understand the reasons why the industry has switched to path tracing *now*, we first need to discuss the many innovative alternative solutions that postponed the wholesale transition until now.

In this chapter we discuss alternative methods for direct visibility (Reyes and ray casting), recursive ray tracing for specular reflections and shadows, and several methods for indirect diffuse (global illumina-

tion). We also discuss dedicated preview renderers. These methods all have particular strengths over path tracing, but the advantage of path tracing is that it is a single method that can do it all quite well.

4.1 Reyes

The Reyes scan-line algorithm [25] splits object surfaces into patches, tessellates each patch into a grid of small polygons ("micropolygons"), computes the color of each grid vertex, and projects the polygon colors onto the screen. This decouples the number of vertex color calculations ("shading"—typically roughly one per pixel) from the number of visibility tests for antialiasing—typically at least 16 per pixel, but often 64 or higher. This decoupling is also great for efficiently computing (approximate) motion blur and depth of field: shading results can be reused by "smearing" them onto multiple pixels. The Reyes algorithm shades an entire grid at a time, which is advantageous for coherent shader execution and coherent access to texture and geometry data. Reyes is also very memory efficient: the image is rendered one tile at a time, and objects are loaded and tessellated on demand only when rendering reaches the image tiles they are in, and deleted as soon as they have been rendered. This allows highly complex objects and densely displaced surfaces. Figure 1.1(a), in the introduction, showed a classic frame from *Toy Story* rendered with Reyes.

Unfortunately, the Reyes algorithm is a bit cumbersome to use since shadows have to be rendered with shadow maps and reflections rendered with reflection maps, both of which need to be generated in a pre-pass. Furthermore, global illumination has to be faked with many manually positioned light sources ("bounce lights")—which requires an expert eye and is very labor intensive.

Over time, the Reyes algorithm has been combined with ray tracing and with many of the global illumination methods described in the following sections.

4.2 Ray casting

An image can be rendered by tracing rays from the eye point through the image pixels to determine what is directly visible in each pixel. A shader is evaluated at every hit point. For reasonable antialiasing at least 16 rays per pixel are required—more if the scene has motion blur or depth of field. This limited version of ray tracing is often referred to as "ray casting".

If the image is divided into tiles with each tile rendered to completion, ray casting has the same desirable data locality and coherency properties as Reyes. However, for progressive rendering, where a single ray is traced through each pixel in each iteration, all geometry and textures are accessed repeatedly, so the entire scene needs to be present in memory to keep iteration times fast. This is problematic in very complex scenes where the tessellated version of the geometry is larger than the available memory.

4.3 Recursive ray tracing

Recursive ray tracing was introduced by Whitted [135] in 1980. At each eye ray intersection point, a shadow ray is traced to each light source, and recursive reflection and refraction rays are spawned. The process repeats at the intersection points of these recursive rays. Ray tracing can render sharp shadows from point and directional lights, and sharp specular reflections and refractions (as seen in for example smooth metal, mirrors, and glass).

For many years, typical commercial ray tracers would tessellate the entire scene up front in order to build an optimal global acceleration data structure. This put ray tracing at a disadvantage compared to Reyes: ray tracing of complex displacement-mapped scenes would immediately run out of memory, whereas Reyes could render them without problems.

On the other hand, an important advantage of ray tracing (and ray casting) is that rendering many copies ("instances") of a few objects is very memory efficient: each instance is simply defined as a transformation of a master object. Rays (and ray hit points) are transformed by

Figure 4.1: (a) Ray-traced glass bottle in a Reyes image from *A Bugs Life*. (b) Combined Reyes and ray tracing image of Luigi from *Cars*. (© 1998, 2006 Pixar/Disney.)

the transformations of the instanced object, but the object itself is not transformed.

The use of ray tracing was very limited in early CG movie work. For example, it was used to render reflections and refractions in a glass bottle in two shots of *A Bugs Life* (1998); see Figure 4.1(a). This was done by having RenderMan (purely a Reyes renderer at the time) call the BMRT ray tracer as a "ray server" [4]. To keep memory overhead down, only a subset of the scene objects was visible to the rays. Later, for the movie *Cars* (2006), a combination of Reyes and full ray tracing was used to render realistic specular reflections and detailed shadows. Figure 4.1(b) shows an image of the car Luigi with ray-traced reflections (most visible in the hood, chrome bumper and hubcaps), and ray-traced shadows. Other early movies rendered partially with ray tracing—using mental ray—were *City of Lost Children* (1995), *Poseidon* (ILM, 2006), and *Speed Racer* (Digital Domain, 2008). More discussion of the strengths of Reyes and ray tracing, respectively, can be found in Christensen et al. [18].

4.4 Distribution ray tracing

Cook et al. [24] introduced distribution ray tracing for stochastic sampling in rendering. Examples are random sampling of shutter time, lens position, and area light sources to render motion blur, depth of field,

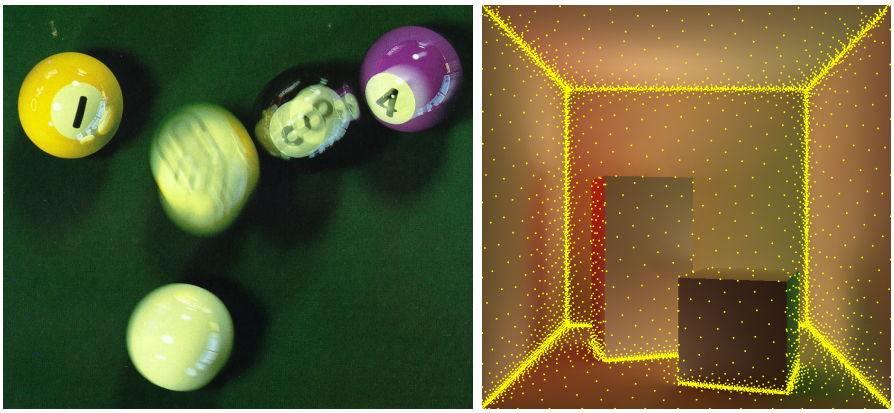

Figure 4.2: (a) '1984' pool balls rendered with distribution ray tracing. (b) Irradiance cache points (marked in yellow) for indirect illumination in a Cornell box.

and soft shadows. Figure 4.2(a) shows a classic distribution ray tracing image of moving pool balls with motion blur and soft shadows.

Distribution ray tracing has also been widely used to compute indirect diffuse (global) illumination. This allowed a drastic reduction in the number of light sources since the indirect illumination no longer had to be faked with manually-placed bounce lights. However, getting results with low noise requires tracing of many rays. In order to reduce the number of rays (and hence increase the efficiency), Ward et al. [133] introduced caching and interpolation of distribution ray tracing results (irradiance) and heuristics to determine where such interpolation is safe. Figure 4.2(b) shows the placement of irradiance sample points and interpolated irradiance results in a Cornell box. The interpolation quality was later improved by utilizing gradients [71, 132, 134] and Hessians [109] of the irradiance results. Tabellion and Lamorlette [118] at DreamWorks introduced a method that first pre-computed direct illumination on all surfaces and stored the results in 2D texture maps (this requires the surfaces to have a uv-parameterization), and then performed one level of distribution ray tracing with lookups in the texture maps at the ray hit points. This technique was first used in the movie *Shrek 2*—as shown in Figure 4.3(a)—and significantly reduced the expense of computing indirect diffuse illumination.

Figure 4.3: Distribution ray tracing images: (a) "Shrek 2" (property of DreamWorks Animation, 2004). (b) "Monsters University" (© 2013 Pixar/Disney).

The use of distribution ray tracing for indirect diffuse illumination was later streamlined by computing and caching the direct illumination on surface patches on demand, thus avoiding the pre-computation pass and the requirement of *uv*-parameterized surfaces [19]. This caching approach also allowed multiple bounces of indirect diffuse illumination to be computed (still without a pre-pass). A combination of Reyes and this technique was used to render the movie *Monsters University*; Figure 4.3(b) shows an example.

Compared with pure path tracing, the sampling is more coherent since it computes direct and indirect illumination over entire surface patches at a time. This amortizes time to analyze complex direct illumination (selecting light sources), and allows reuse of (interpolated) cached values for subsequent rays hitting that patch. Furthermore, distribution ray tracing from a point can stratify the hemisphere directions, whereas for path tracing of e.g. camera rays from a pixel to a curved diffuse surface, each camera ray hits the surface at a position with a different surface normal, and the hemisphere strata are rotated and overlap; hence there is less benefit from stratification and more noise for the same number of diffuse reflection rays. This difference is illustrated in Figure 4.4. (The same issue happens for quasi-random samples where the stratification is implicit.)

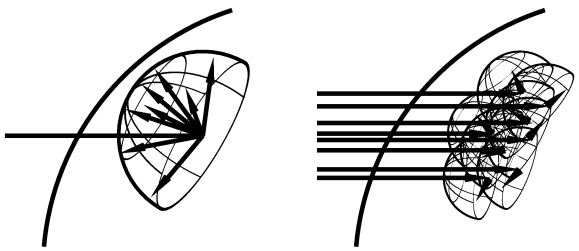

Figure 4.4: Hemisphere stratification for diffuse reflection: (a) Distribution ray tracing. (b) Pure path tracing.

On the other hand, the caching approach uses more memory and cannot benefit from object instancing: even though the geometry can be instanced, the illumination on each object instance is different, so each object needs separate cache values no matter whether it is instanced or not. In contrast, path tracing, which doesn't cache illumination, can take full advantage of object instancing.

4.5 Photon mapping

The photon mapping method [51, 52] is a three-pass method to compute indirect illumination (including indirect diffuse illumination and caustics):

1. Emit light particles (photons) from the light sources, trace them through the scene, and store them in a point cloud when they hit a surface (or get scattered in a volume).

2. Organize the photon point cloud into a global data structure called a photon map—typically a kd-tree—which is independent of surface complexity and parameterization.

3. Render using local photon density and power to determine indirect illumination.

Figure 4.5 shows the familiar box with all illumination on diffuse surfaces determined from a photon map. The image shows the "stucco-

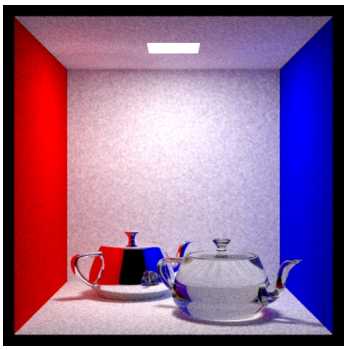

Figure 4.5: Photon map irradiance estimates.

looking" low-frequency noise that is characteristic for this kind of density estimation. Another problem with such density estimates is that they tend to be too dark near surface edges—although this can be avoided by using a convex hull radiance estimate [55]. To reduce the low-frequency noise it is common to calculate the direct illumination separately, and only use the photon map for indirect illumination. To further reduce the low-frequency noise, another common variation is to change the rendering step to compute both direct illumination and caustics separately, and do one level of distribution ray tracing (with irradiance caching and gradients [132]) to compute indirect diffuse illumination. This rendering method is often called photon mapping with "final gather" [51].

Huge photon maps can be handled with the radiosity atlas method [15], and progressive versions of photon mapping were introduced by Hachisuka et al. [39] and Knaus and Zwicker [65].

Photon mapping has been used for very specialized effects in a few movies. It was used to render a caustic from a whisky glass in *Final Fantasy: The Spirits Within* (2001)—this was rendered with a plug-in to the Maya renderer (and composited on top of RenderMan Reyes images). Photon mapping was also used for rendering artistic photon beam effects [50, 88] in volumes in Disney's *Tangled* and *Frozen*—as shown in Figure 4.6.

Figure 4.6: Photon beams from *Tangled* and *Frozen* (© 2010, 2013 Disney).

Even though photon mapping was built into renderers such as mental ray and RenderMan, it was never very successful as a general global illumination solution for movie production. We can only speculate why it never "took off", but one possible explanation is that users found it hard to tune its parameters: how many photons should be emitted, how many photons should be used for density estimation, and how many final gather rays should be used?

Photon mapping has recently been incorporated into advanced rendering techniques such as VCM/UPS (see Section 8.3), so photon mapping is probably—indirectly—still part of the future of movie rendering.

4.6 Point-based global illumination

Point-based global illumination [9, 14] is also a three-pass method for computing indirect illumination:

1. Generate a point cloud representation of reflected direct illumination on all surfaces in the scene. Each point represents the color and geometry of a tiny part of a surface (often called a surface element or "surfel").

2. Organize the point cloud into a cluster hierarchy, with total directional illumination from clusters represented as spherical harmonics.

3. Render the image, with indirect illumination computed from the hierarchical point cloud. At each surface point, gather illumination from nearby points and distant clusters in the point cloud. Rasterize the points and clusters into a very coarse raster (for example a cube with 12×12 pixels on each side) and accumulate this illumination weighted by the surface reflectance function evaluated for each raster pixel.

Figure 4.7(a) and (b) show examples of point clouds used to render the movie *Up* (Pixar, 2009), and Figure 4.7(c) shows the resulting point-based global illumination. Figure 4.7(d) shows another example of point-based global illumination: Davy Jones from *Pirates of the Caribbean 2: Dead Man's Chest* (ILM, 2006).

Figure 4.7: Point-based global illumination: (a) Point cloud for key light. (b) Point cloud for fill lights. (c) *Up* living room rendered with indirect illumination from the two point clouds (© 2009 Pixar/Disney). (d) Davy Jones from *Pirates 2* (© 2006 Disney and Jerry Bruckheimer).

The biggest advantages of the point-based method are that it is memory efficient (the working set is small since hierarchical point clouds can be cached) and the computed indirect illumination has no noise. The disadvantage is that the point clouds must be generated in a pre-pass, so the method is not suitable for progressive rendering.

Ritschel et al. [104] presented a GPU version of the algorithm. Their version rasterizes the point clouds (surfels) onto a single raster (instead of a cube), with directions being transformed to raster pixels by an importance-warping function based on the surface reflectance function. Kontkanen et al. [69] developed an extension where point clouds are kept out of core, and only read in on demand, thus allowing the rendering of even more complex scenes. Figure 4.8 shows an example of out-of-core point-based global illumination in a very complex scene from *Kung Fu Panda 2* (DreamWorks, 2011). Point-based global illumination has been used in more than sixty movies, but is currently being phased out in the industry in favor of single-pass rendering methods.

Figure 4.8: Point-based global illumination from *Kung Fu Panda 2*. (Property of DreamWorks Animation, 2011.)

4.7 Preview renderers

Several renderers have been developed specifically for quick previews during lighting setup. This is usually done by having a "deep framebuffer" representing the geometry in each pixel [106]. The deep frame-

buffer enables quick updates since screen visibility has already been resolved, but restricts the approach to static geometry.

The Lpics renderer [93] used a GPU to execute simplified versions of shaders. It recomputed illumination and shadows for each light individually if there was a change in that light's intensity or other parameters. The shaders were manually translated from RenderMan shading language (RSL) to GPU-executable kernels; the manual translation made it cumbersome to update when the RSL shaders changed.

The Lightspeed system [100] was more general. It handled transparency, motion blur, depth of field, and subsurface scattering. A second generation of Lightspeed was even more interactive, handled indirect illumination, and even allowed some moving geometry (shadow casters and bounce cards). Both versions of Lightspeed had automatic shader translation from RSL to the Cg language for GPUs, thus avoiding the manual shader translation step. The automatic translator had to be updated, however, when the RSL syntax or language capabilities were extended.

Lpics and Lightspeed were used for a few years at their respective studios, but then fell out of use. While these dedicated systems enabled a limited form of preview for Reyes-based rendering, it is probably fair to say that in practice, it is more convenient to have a single renderer for both previews and final frames, and a single set of shaders to maintain.

5

Advanced Path Tracing

Basic path tracing is simple to implement, but creates noisy images that converge slowly. Many techniques have been developed to make path tracing faster: both algorithmic improvements and better use of modern many-core hardware. Furthermore, path tracing has been extended from surfaces to also render important cases like hair, fur, volumes, and subsurface scattering, and has been made more flexible to allow selection of specific light paths and non-physical effects.

5.1 Algorithmic improvements

The algorithmic improvements include better sampling techniques and sample patterns, use of ray differentials to determine texture filter sizes and geometry tessellations, stochastically choosing between thousands of light sources, and ray (and ray hit) reordering for improved coherency.

5.1.1 Improved sampling techniques

Many general techniques have been developed to improve the accuracy and efficiency of Monte Carlo simulation—see for example the classic

textbooks by Kalos and Whitlock [59], and Spanier and Gelbard [114]. These techniques include importance sampling (where knowledge of the sampled function is used to place the sample points), control variates (where we estimate the difference between our target function and a similar function with a known integral), and Russian roulette (probabilistically terminate low-contribution paths without introducing bias).

In computer graphics, we use the same techniques to reduce image noise [5]. Also, a clever way to combine the results of multiple importance sampling techniques has been developed for computer graphics by Veach and Guibas [121]. This combination is widely used; one of many uses is combining importance sampling according to the surface reflectance function with importance sampling according to the light source position: either approach alone gives low sample noise for some surface areas but not for others; combining them with multiple importance sampling yields the best of both, as illustrated in Figure 5.1.

Computing the incident illumination to a surface point can be formulated as an integration. We commonly split the integration domain into light sources vs. other (non-emissive) surfaces. Efficient sampling of direct illumination from an area light has been covered by e.g. Shirley et al. [111], Ramamoorthi et al. [101], and Subr et al. [115]. More examples of the use of efficient sampling techniques in rendering can be found in the course notes by Hery and Villemin [46]. We will discuss efficient sampling of many lights in Section 5.1.4.

Some very promising importance sampling techniques have recently been developed specifically for path tracing: "hero" wavelengths [99, 136] can be used for efficient spectral rendering (i.e. simulating more wavelengths than the usual red, green, and blue components), and advanced next-event estimation [42] can reduce noise in caustics. The images in Figure 5.2 illustrate these two techniques; both images were rendered with Weta's in-house renderer Manuka. There has also been a wealth of new sampling techniques developed specifically for volumes, which we discuss in Section 5.3.2.

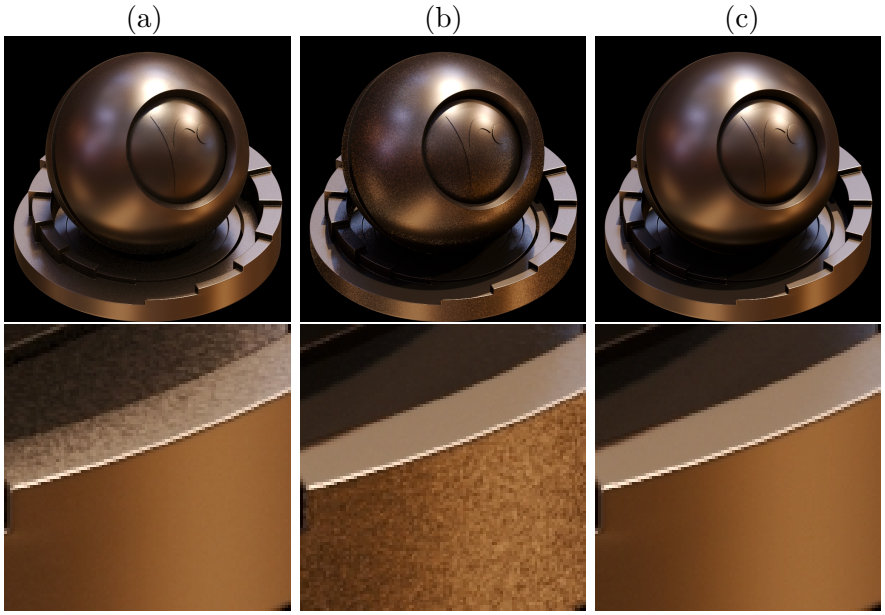

Figure 5.1: Multiple importance sampling. Top row: (a) Importance sampling according to the surface reflectance function. (b) Importance sampling according to the light source position. (c) Multiple importance sampling according to both surface reflectance and light source. Bottom row: close-ups. (Images courtesy of Christophe Hery.)

Figure 5.2: Recent techniques for improved sampling: (a) Hero wavelengths. (b) Manifold next-event estimation. (© 2014, 2015 Weta Digital.)

5.1.2 High-quality sample patterns

One of the most "dry" topics that must be considered for efficient rendering is the use of high-quality sampling patterns. Using good sample patterns can reduce noise and improve convergence very significantly. An ideal sample pattern has points that are well distributed, i.e. no points very close to each other and no large areas with no samples. Selecting sample patterns is a trade-off between conflicting goals, and no single pattern is ideal for all uses: for example, for progressive rendering or adaptive sampling, different patterns should be selected than for non-progressive, non-adaptive rendering, and the optimal strategy for high-dimensional sampling is not clear-cut.

The simplest sample pattern is uniform random samples (generated with e.g. a linear congruential function [66] or the Mersenne twister [82]). Random samples are simple to generate and use, but as shown in Figure 5.3(a) they tend to clump together and leave gaps in the sample domain. This gives slow convergence. A much more even distribution of sample points can be created by generating a number of candidate points for each new sample, and picking the one that has the largest distance to the previous points [86]. This gives a pattern where the points have nice even distances and there are no large gaps, as shown in Figure 5.3(b). However, even though the samples look nicely distributed, unevenness in the distribution remains. For example, for 500 best-candidate samples, there are typically anywhere from 115 to 135 samples in each quadrant (whereas we would expect very close to 125 samples if the distribution was even).

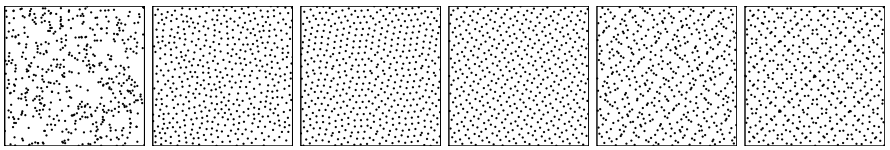

Figure 5.3: 500 two-dimensional samples from six sample patterns: (a) Uniform random. (b) Best candidates. (c) Correlated multijitter. (d) Larcher-Pillichshammer. (e) Halton. (f) Sobol.

If we know a priori how many samples are needed, we can use sample *sets* where the order of the sample points does not matter. Three popular stochastic sets are jittered [23], multijittered [13], and correlated multijittered [62] samples. To generate N jittered samples, the domain is divided into $\sqrt{N} \times \sqrt{N}$ cells, and a random sample point is placed in each cell. Multijittered samples furthermore ensure that the sample points fall into N rows and N columns with 1 sample each. Correlated multijittered samples furthermore have a larger average distance between the sample points. Figure 5.3(c) shows a correlated multijittered sample set. Two other popular sample sets are the Hammersley and Larcher-Pillichshammer quasi-random sets. The samples in both sets have x component i/N and a y component that is computed with simple formulas [68]. Figure 5.3(d) shows a Larcher-Pillichshammer set with 500 samples. Out of these sample sets, Larcher-Pillichshammer and correlated multijittered samples give the best results.

Sample sets are only useful if all the samples are used, so they are not suitable for adaptive sampling or progressive rendering. For progressive rendering where the images are displayed while being rendered, it is not sufficient that the final image has low noise—we also want the partially computed images to have as low noise as possible. For this purpose we need ordered sample patterns where each prefix of the sequence is well distributed. We call such patterns *sequences*. Two popular quasi-random sample sequences are Halton and Sobol sequences, with the first 500 samples of each sequence shown in Figure 5.3(e) and (f). As the figure shows, some of the sample points are very close to each other, and the Sobol sequence has pronounced diagonals which can lead to aliasing. Nonetheless, for progressive rendering or adaptive sampling, sequences are preferrable over sets.

If a quasi-random pattern is used, there is a potential problem in that there is only *one* pattern of each type: if the same pattern is used in all pixels, very visible aliasing will occur, especially at low sample counts. To avoid this, the pattern can be offset by a different toroidal shift (Cranley-Patterson rotation [26]) in each pixel, or the digits of the sample points can be scrambled [89, 68, 96] by a different permutation (which is a simple bit-wise xor in some cases) in each pixel.

The previous discussion was focused on two-dimensional samples, but for rendering realistic images we need samples in higher dimensions. Some patterns—for example Hammersley, Halton, and Sobol—have straightforward extensions to higher dimensions. However, caution should be used when using these quasi-random sequences in high dimensions since they have unfortunate systematic patterns in some dimensions. For example, Figure 5.4(a) shows the first 100 samples from the Halton sequence in dimensions 6 and 7 (base is the primes 17 and 19). There are two large gaps in the sample domain with no sample points at all, and the samples have a very regular pattern making them prone to aliasing. As another example, Figure 5.4(b) shows the first 500 samples from a Sobol sequence [56, 36] in dimensions 14 and 15. The samples are clumping together and roughly half the domain has no samples at all. Only if more samples are taken will the gaps fill in. Digit scrambling helps in both cases, but the distribution is still not as even as in lower dimensions. For best-candidate samples, if they are generated in high dimensions, it is important that they are generated in such a way that their lower-dimensional projections also have a good distribution [86, 102].

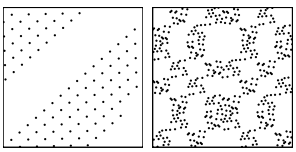

Figure 5.4: Initial samples in high dimensions: (a) 100 Halton samples in dimensions 6 and 7. (b) 500 Sobol samples in dimensions 14 and 15.

Alternatively, independent 2D sets or sequences can be combined; this is is referred to as "independent sampling" or "padding" [110, 68], or one can switch to random samples beyond a certain dimension [48, 75]. If sample *sets* are combined, the order of the samples in all but the lowest dimensions have to be shuffled to avoid systematic correlation. For sample *sequences*, shuffling the order of the entire sequence will severely reduce the quality of the sample pattern, so the shuffling has to be done within groups of samples.

5.1.3 Ray differentials

A ray is defined mathematically by an origin and a direction. Igehy [47] introduced the concept of ray differentials; a ray differential expresses how the origin and direction would differ—through propagation and specular reflections and refractions—for a ray emitted from a slightly different origin or direction. Conceptually, one can think of this as tracking where a slightly different "neighbor" ray would travel—with modest overhead since no extra rays are actually traced. Igehy's ray differentials provide an elegant way to determine optimal texture filter sizes for specularly reflected and refracted rays.

Christensen et al. [18] extended ray differentials to diffuse reflection in the context of distribution ray tracing, and used the ray differentials to select geometry tessellation and texture detail. Eye rays and specular reflection rays from flat surfaces are coherent, while reflection rays from curved or diffuse surfaces are incoherent. The incoherent rays can thrash geometry and texture caches if the finest resolution is always chosen. Fortunately, less geometric and texture detail is necessary in the incoherent cases; this observation was formalized by analyzing the ray differentials for different types of scattering. The authors utilized this observation to obtain efficient multiresolution caching of geometry and textures (including displacement maps) for complex scenes. Figure 5.5 shows five different tessellations of a surface patch; in this example the finest tessellation rate is 14×11 and the coarsest tessellation is simply the four corners of the patch. One can think of the various levels of tessellation as a MIP map [137] of tessellated geometry; the ray differential is used to select the appropriate tessellation.

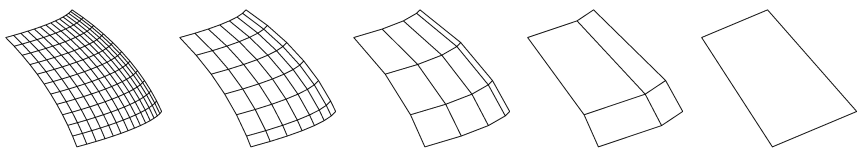

Figure 5.5: Multiresolution tessellation example for a surface patch: 14×11 quads, 7×6 quads, 4×3 quads, 2×2 quads, and 1 quad.

The same observations about ray coherency and ray differentials also apply to path tracing. Since many paths are traced for each pixel, the differentials do not have to be as accurate as the ones derived by Igehy. In fact, the differentials can be simplified to just two isotropic quantities per ray: the radius of the ray at its origin, and the ray spread which expresses how much wider the radius gets for each unit the ray travels. Figure 5.6 shows examples of path differentials at specular and diffuse reflections.

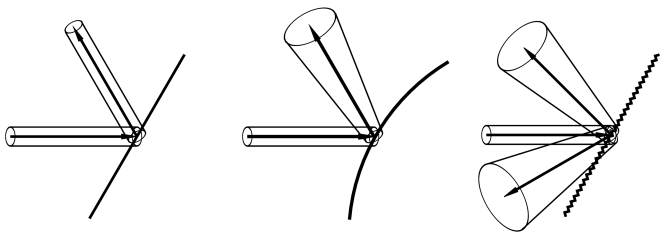

Figure 5.6: Path differentials: (a) Specular reflection from a flat smooth surface. (b) Specular reflection from a curved smooth surface. (c) Diffuse reflection from a rough surface.

Suykens and Willems [116] presented two strategies for calculating path differentials. Recent work on covariance tracing [7, 8] has further formalized and refined the calculation of optimal filtering sizes for path tracing.

Yet another use of ray differentials is to switch from (expensive) subsurface scattering simulation to (faster) diffuse reflection when the diffuse mean free path of the subsurface scattering is shorter than the ray differential radius at a shading point.

To summarize, ray (path) differentials are essential to determine the proper geometry tessellation level and texture filter sizes, thereby avoiding geometry and texture cache thrashing and unnecessary variance in the texture lookup results (if too narrow filter sizes were used) or too coarse geometry and overly blurry textures (if too wide filter sizes were used).

5.1.4 Direct illumination from many lights

It turns out that, ironically, one of the unexpectedly hard problems in path-traced global illumination is computing the direct illumination accurately and efficiently—particularly in scenes with thousands or millions of light sources. For efficiency, only a small fraction of the lights can be sampled at each surface point.

Ward [131] and Shirley et al. [111] did early work on this problem. They utilized the fact that even in scenes with thousands or millions of lights, only a few light sources will create strong illumination in each part of the scene. With Ward's method, the lights are sorted according to their potential contribution to a point, and only the lights above a specified threshold are shadow-tested. The illumination from the remaining lights is estimated by taking into account the average occlusion per light and per surface point. The approximation tolerance can be increased for paths with low contribution to the image. Shirley's method divides the scene into regions, and for each region the light sources are divided into (locally) bright and dim. The bright lights are sampled carefully, but only one or a few of the dim lights are chosen stochastically and sampled (with their contribution weighted higher, according to their probability of being chosen).

More recent work clusters the light sources and sorts their potential contributions at each receiving point, or stochastically chooses a radius for each light source beyond which it is ignored, as described by for example Tokuyoshi and Harada [119]. Many of these ideas are conceptually similar to the hierarchical clustering utilized by the point-based global illumination methods described in Section 4.6 and many-light methods [27] such as variants of Lightcuts [128, 129, 130]. Learning algorithms [124] can be applied too: for example, if in some parts of a scene a nearby, bright light is found to be entirely occluded, this information can be utilized to reduce the probability of sampling it there. But changing probabilities has the unfortunate side-effect that sample-sequence stratification shifts during progressive rendering, ruining stratification and hence introducing more sampling noise in the illumination.

5.1.5 Ray reordering and sorting

Rays can be reordered to increase coherency of geometry and texture accesses. In the Toro renderer [95, 97], rays are queued, and once sufficiently many rays are waiting to be intersection-tested against an object, that object is read in, tessellated, and (optionally) displacement mapped. This reordering makes it possible to render scenes that are too large to fit in memory.

Disney's in-house renderer Hyperion [31] sorts both rays and ray hits. It collects rays into batches of 30–60 million, where each batch is first sorted into bins on local solid-state disk (SSD) based on the six cardinal directions. Each bin is then sorted based on ray origin positions until groups of 4096 rays have been obtained, and then sorted further based on ray directions until ray groups of 64 rays have been found. Each group of 64 rays forms a coherent ray packet.

After the rays have been traced, the ray hits are sorted based on which texture file they need to read from. This makes texture accesses coherent and minimizes the number of times each texture file is reopened—this is essential for complex geometry since the ptex texture format does not have MIP map levels covering multiple faces. Each texture file is opened only once for each ray batch. Figure 5.7 shows a frame from *Big Hero 6* rendered with Hyperion.

Figure 5.7: The city of San Fransokyo in *Big Hero 6* (© 2014 Disney).

Relying on large batches of rays for efficiency is not ideal for interactive rendering where each iteration typically only traces one eye ray per pixel (around 2 million eye rays). Áfra et al. [1] recently described coherent shading with much smaller ray batches.

5.2 Hardware efficiency and parallel execution

Modern processors have many computation cores and wide instruction sets.

5.2.1 SIMD ray tracing: intersection tests and BVH traversal

Each new generation of computers has more compute power and memory than its predecessor. However, exploiting the new and faster multicore and many-core processors requires careful multithreading and judicious use of SIMD (single instruction multiple data) instructions such as SSE and AVX.

Intel added 4-wide SIMD instructions to their CPUs in 1999. These instructions run identical operations (add, multiply, square root, etc.) on sets of four adjacent float data in a single clock cycle. Early work on speeding up ray intersections using these new instructions was done by Wald and colleagues [125, 126]. Their method intersection tested four rays against one triangle in parallel, which is efficient for coherent rays such as eye rays from adjacent pixels or shadow rays from adjacent eye ray hit points. An alternative that works better for incoherent rays is to intersection test one ray against four triangles in parallel [17]. (Modern variations use a hybrid approach: trace bundles of rays when the rays are coherent, single rays when incoherent.) Traversal of a ray acceleration data structure can also be sped up using SIMD instructions. A bounding volume hierarchy (BVH) can be traversed efficiently by storing four bounding boxes together and intersection testing them in parallel [126]. Other significant speedups came from carefully arranging the ray and geometry data to align on memory cache lines. These days we have wider SIMD instructions: AVX can perform 8 or 16 intersection tests simultaneously, enabling further speedups.

GPUs have higher bandwidth than CPUs but also memory with high latency. There are typically 32 threads running together in a "warp", and those threads must execute the same control flow (although, for example, CUDA presents a MIMD-like software programming model). This means that we need more rays to be traced together with the same control flow (since control flow divergence is detrimental to performance). Aila and collaborators have explored ray tracing efficiency on GPUs [2, 75]. They found that the most efficient strategy is to use persistent threads, i.e. the same kernel running in any given thread but have the kernel fetch work (e.g. rays to trace) from a global pool until the pool is empty. A wavefront of rays is traced, then the surface shaders at the ray hit points are executed, a new wavefront of rays is generated and traced, and so on.

These days, the major microprocessor vendors provide basic libraries for efficient ray tracing on CPUs and GPUs—for example Embree [127] and OptiX [90]. These libraries are very convenient for renderer developers; using these building blocks ensures good performance without having to rewrite highly specialized ray tracing code for each new generation of hardware.

5.2.2 Many-core scalability

Ray tracing and path tracing are sometimes called "embarrasingly parallel" in the sense that it is "trivial" to obtain near-perfect parallel speed-ups. It is true that each pixel color can be computed completely independent of other pixels. In theory all pixel colors could be computed simultaneously if the entire scene (all tessellated geometry, textures, etc.) is pre-loaded into memory.

However, in reality it is not that simple. Caching is necessary if the geometry and textures do not fit in memory, and there may be cache contention and many threads waiting for the same data to load. Even if all the data is in main memory, if it is not in the L1 cache of the processor that needs it, a delay will occur. Incoherent accesses tend to quickly push data out of L1 processor caches. Adding more processors will only make matters worse if memory isn't scaled up at the same time. There might be no speedup at all beyond a certain number of

processors if they all are waiting for different data to be loaded from off-chip memory with a fixed bandwidth. Another dangerous trap is if an algorithm is designed such that many processors try to read and write the same data simultaneously (protected by a lock): then adding more processors might actually slow the execution down. Getting optimal use out of 64 or more processors requires suitable algorithms and thoughtful software design and implementation.

5.3 Beyond surfaces

Realistic movie scenes do not consist of only surfaces, but often also contain hair, fur, and volumes, and have light scattering below the surfaces.

5.3.1 Hair and fur

Early work on rendering of fur was done for *Bunny* (1998) and the furry monsters in *Monsters, Inc* (2001). One trick is to not model individual hairs, but render planes with textures of hair (and transparent space between the hairs). Another trick is to widen the hairs, but at the same time make them more transparent; this reduces sampling noise. Such tricks are less suitable for physically realistic path tracing, though.

Hair and fur consist of very dense geometry, so they require highly efficient ray intersection algorithms. A recent paper describes an acceleration data structure developed for fast ray tracing of hair and fur [139]: tight bounding boxes that are aligned with the local hair orientation instead of loose axis-aligned bounding boxes.

In order to make hair and fur look realistic, it is important to have good algorithms for sampling and shading evaluation. The Marschner model [81] is the most popular hair scattering model. Efficient sampling methods have been developed by Hery and Ramamoorthi [45], d'Eon et al. [29], Pekelis et al. [92], Yan et al. [142], and Chiang et al. [11]. Figure 5.8 shows fur in two CG movies: a rabbit from *Alice in Wonderland* (2010) rendered using Arnold and a lamb from *Zootopia* (2016) rendered using Hyperion.

Figure 5.8: Path-traced fur: (a) Rabbit from *Alice in Wonderland*. (b) Lamb from *Zootopia*. (© 2010, 2016 Disney.)

5.3.2 Volumes

In the early days of computer graphics in movies, rendering of just surfaces was already pushing against the limit of what was computationally tractable, so fully-fledged, physically based volumetric rendering was simply beyond reach. Consequently, volumetric effects were either shoe-horned into production surface renderers (in an often cumbersome fashion) or generated in a separate volumetric rendering pass (often in a different renderer) for later compositing. Both of these options complicated the production workflow and made simulating light transport between surfaces and volumes difficult. The computational complexity also often meant that only single scattering (direct illumination) was computed in the volumes.

Early examples of volumetric effects in movies relied on ray marching [3] to compute direct illumination from light sources. Hanson's chapter in the book by Apodaca and Gritz [4] describes how this approach was applied in the movie *Contact* (1997) within a Reyes framework, and Wrenninge [140] provides a more recent treatment. Deep shadow maps [78] were introduced to provide the benefits (and drawbacks) of shadow maps in the context of volumetric rendering. Several production renderers (including Houdini's Mantra and Pixar's RenderMan) generalized Reyes-style micropolygon rendering to micro*voxel* [20] rendering, where volumes are first divided into small vox-

els which are then shaded before being sampled on screen. More recent work by Wrenninge [141] has also examined ways to perform correct motion blur in this microvoxel framework. While microvoxels elevate volumes to equal status as surfaces, the approach suffers from the same basic limitations that Reyes has for surfaces, making global illumination and progressive updates challenging.

One major benefit of moving to path tracing is that rendering of volumetric effects (e.g. fog, clouds, smoke) fits naturally within the same framework as surface rendering. The path tracing algorithm, in essence, remains unchanged. We still need to construct random paths between the light sources and the eye, but these paths can now have vertices not only on solid surfaces, but also within the volumetric media (see Figure 5.9). This can have a huge impact on the ease of incorporating such effects in a production environment.

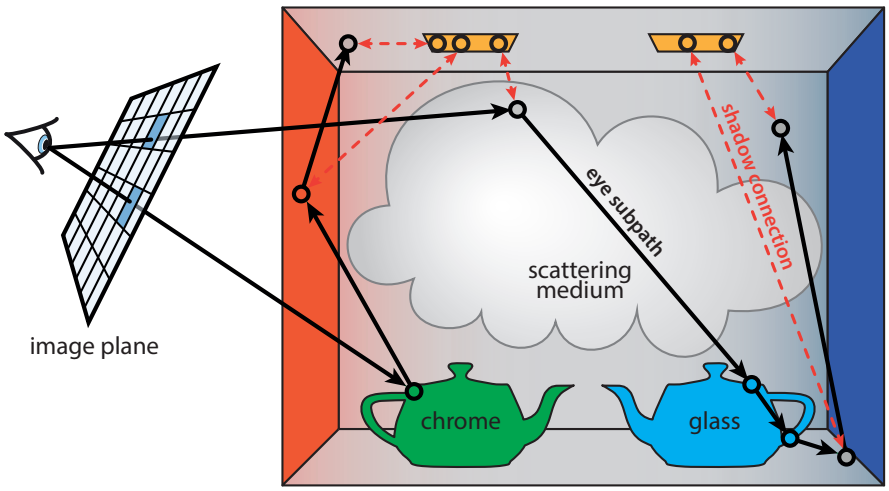

Figure 5.9: An illustration of tracing paths from the eye to the light sources in a Cornell box scene with two teapots and a scattering medium.

In a scene consisting of only solid surfaces in a vacuum, rays travel unobstructed and only scatter at their intersection locations with surface geometry. Extending path tracing to volumetric path tracing [74] requires two changes to the base algorithm: we need 1) a way to evaluate the potential contribution of a light source to a path vertex in the

Figure 5.10: Images rendered with volume path tracing: (a) Residual ratio tracking (© 2014 Disney). (b) Equiangular sampling (© 2012 Sony Pictures Imageworks, Inc. All rights reserved).

presence of volumes, and 2) a way to generate light paths with vertices within the volume.

When evaluating shadow rays in the presence of media, the binary visibility function (which previously took on the value of *either* 0 or 1 for opaque surfaces), generalizes to the transmittance function (which can take on any fractional visibility value *between* 0 and 1). Accurately and efficiently evaluating this fractional visibility therefore becomes an important consideration in volumetric path tracing. Novák et al. [87] proposed residual ratio tracking, which accelerates this computation. Figure 5.10(a) shows an image from Disney's *Big Hero 6* (2014) which leveraged residual ratio tracking for efficiently computing the fractional visibility in the thick fog. When performing shadow connections, the strategy for sampling locations on the light source can also impact noise considerably: Villemin and Hery [123] developed an approach for sampling volumetric light sources directly.

In volumes, scattering can occur at any location along a ray, so the distance must be sampled probabilistically. One strategy is to compute a so-called *free-flight distance* within the medium, which corresponds to importance sampling the transmittance term. This distance can be computed by incremental ray marching [3, 94, 53, 37], or by a technique called Woodcock tracking [22, 32, 112, 138], which was introduced to graphics by Raab et al. [98]. Yue et al. [143] and Szirmay-Kalos

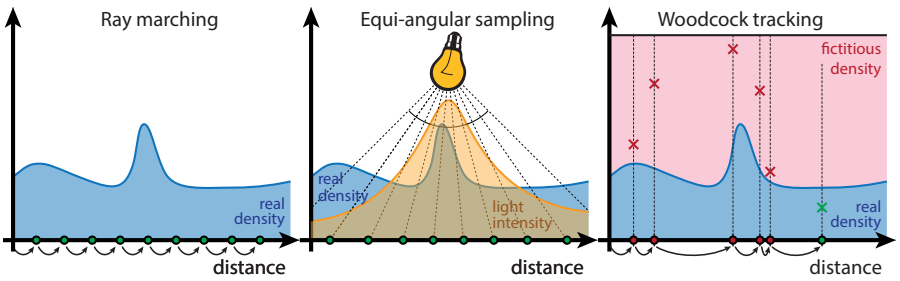

Figure 5.11: Uniform ray marching (left) marches at regular intervals along the ray. Equiangular sampling (middle) samples proportional to the inverse-squared distance term of the lighting. Woodcock tracking fills the medium with fictitious density to create a homogeneous medium, and then proposes tentative free-flight distances within this denser medium, rejecting them until it probabilistically collides with the real density.

et al. [117] proposed ways to accelerate Woodcock tracking for highly heterogeneous volumes. Kulla and Fajardo [70] proposed equiangular sampling—a technique previously developed in the neutron transport field [58, 103]—to account for the inverse-squared distance term from light sources when sampling the scattering distance along rays. This strategy can dramatically reduce noise in single scattering for scenes containing light sources within volumetric media (see Figure 5.10(b)). Georgiev et al. [34] extended this idea to importance sample a chain of path vertices while accounting for both the inverse-squared distance term and scattering phase function. Figure 5.11 illustrates sampling the distance using three different sampling strategies.

5.3.3 Subsurface scattering

Realistic modeling of subsurface scattering is important for rendering believable images of translucent materials such as skin, meat, fruits, plants, wax, marble, jade, milk and juice. A common approach is to consider subsurface scattering under a flat surface after dozens or hundreds of bounces—a so-called diffusion model. Computer graphics researchers have developed increasingly sophisticated and accurate physically based diffusion models from the simple dipole diffusion model [54] to the quantized diffusion [28] and photon beam dif-

fusion models [38]. A simple but accurate approximation of all subsurface bounces—including single-scattering and diffusion—has been developed as well [10, 16].

The first practical approaches to rendering of subsurface scattering were point-based, and later based on distribution ray tracing. More recently, subsurface scattering diffusion models have been shoe-horned into the path-tracing framework (see for example King et al. [64] or similar techniques), further making path tracing a unified, general rendering solution. The subsurface scattering is computed by selecting a random position on the surface, evaluating the incident illumination there, and multiplying by the diffusion profile. For more efficient sampling, the diffusion profile is used to importance sample the position on the surface. Figure 5.12(a) shows a highly realistic, carefully recreated young Schwarzenegger from the movie *Terminator: Genisys* (2015); he was rendered with path-traced subsurface scattering in RenderMan.

Figure 5.12: Path-traced subsurface scattering: (a) Diffusion model used in rendering a young Terminator (© 2015 Moving Picture Company). (b) Brute-force approach on the snow monster from *Frozen Fever* (© 2015 Disney).

An alternative method that has only recently become viable for subsurface scattering is brute-force volume path tracing (as described in the previous section, but for many bounces in a very dense volume). This approach does not assume a flat surface, and can correctly ren-

der scattering in crevasses and on thin or granular objects [10, 12]. It requires simulation of many bounces to get a realistic result, so it is slower than using a diffusion model. Recent research proposes better sampling by steering paths towards the surface where the brightest illumination comes from [83], which can make the brute-force approach more efficient. Figure 5.12(b) shows the snow monster from *Frozen Fever* rendered with brute-force volume path tracing using Disney's Hyperion renderer.

5.4 Flexibility

Allowing the user to separate and artistically manipulate light paths has become a practical and convenient tool for path tracing in movie production.

5.4.1 Light path expressions

Light path expressions are very useful to distinguish between the different paths that light can take from the light sources to the eye. This can be utilized to increase or decrease the intensity of certain light paths, or even delete some paths entirely. For example, caustic paths might contribute a lot of noise to a path-traced image without any desirable visible contribution, so they are sometimes explicitly omitted from the final images. Separate images for diffuse and specular reflections are also needed as input to denoising algorithms (more on this in Section 6.2).

Light path expressions can be written concisely using Heckbert's regular expression notation [44]: E denotes the eye (camera), L denotes a light source, D is diffuse reflection, and S is specular reflection. '|' is a choice between two paths, '*' means zero or more repeats, '+' means one or more repeats, and '[x]' means x is optional. With this notation, light directly from the light source to the eye is LE, light reflected exactly once (i.e. direct illumination) is $L(D|S)E$, light reflected at least twice (i.e. indirect illumination) is $L(D|S)(D|S)^+E$, and light reflected any number of times (all illumination) is $L(D|S)^*E$. Figures 2.1 and 2.2 were generated using light path expressions.

Extensions of this notation allow distinction between reflection and transmission, and between light sources and emissive geometry. In ray tracing and path tracing settings it is common to reverse the direction of the expressions, i.e. starting at E and ending at L.

5.4.2 Artistic manipulation

It is possible to take many of the same liberties in path tracing as were taken with classic rendering techniques [6]: some lights might only illuminate certain objects, some objects may only cast shadows on certain other objects, some ray paths can be explicitly omitted, reflection rays can be "bent", some light paths can be enhanced or reduced, color bleeding from one particular object to another can be increased or decreased, caustics and secondary specular reflections can be deleted or brightened, etc. A recent paper by Schmidt et al. [107] formalizes and proposes a practical system for several such manipulations of light transport paths, as shown in Figure 5.13. The survey by Schmidt et al. [108] provides a more exhaustive overview of available techniques for artistic editing of appearance, lighting, or material.

Figure 5.13: The framework by Schmidt et al. allows modifying the light transport of the scene (left) by removing the caustics caused by the car, re-directing sunlight passing through the windows, and altering the reflections in the mirror (middle). Their interactive global illumination preview tool (right) suggests and visualizes bundles of paths flowing through a selected region to aid manipulation. (Image from Schmidt et al. [107]; used with permission.)

6

Enabling Technology

In this section we discuss two developments that have greatly helped making path tracing a viable rendering technique for movies: physically based rendering and denoising.

6.1 Physically based rendering

A prerequisite for path tracing is that physically realistic rendering is acceptable. Superficially, there might seem to be an inherent conflict between targeting physical realism and "art directability", i.e. the ability to change every aspect of the light in the scene at the director's whim. However, it is very helpful to start with a physically based image and then make targeted changes, instead of spending a lot of effort trying to obtain a reasonably realistic look. Physically based rendering simplifies VFX work to match rendered elements to real images and raises the bar of visual quality. It gives richer, more plausible, consistent, and predictable results. A big effort to use realistic lights and surface materials was made at ILM [113]; an example is shown in Figure 6.1. Unlike more ad-hoc approaches, physically-based energy conserving shading models hold up under a variety of lighting environments. The separation of

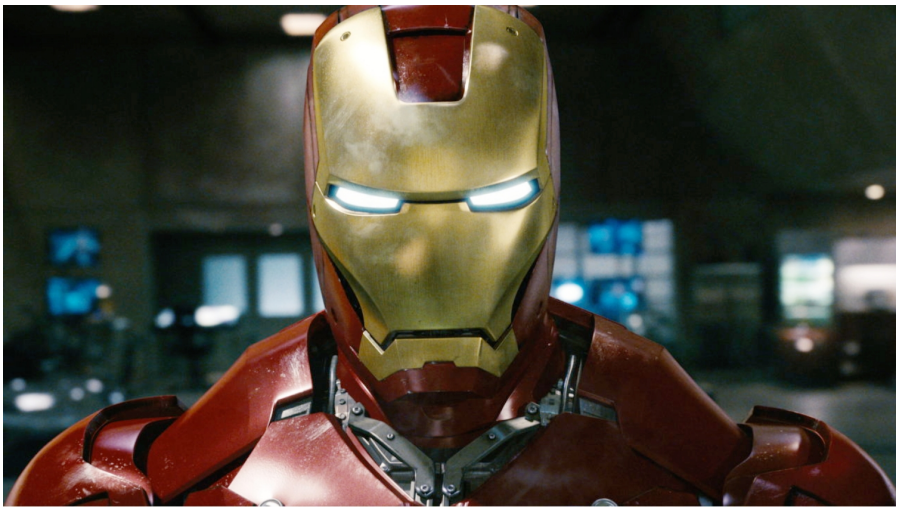

Figure 6.1: *Iron Man* with realistic materials illuminated by high dynamic range illumination. (© 2008 Marvel Entertainment.)

material description and rendering algorithm also gives more portable assets and less scene-specific tweaking. The elimination of the need to manually place fake "fill" or "bounce" lights to mimic the effects of indirect diffuse illumination means less work to set up every scene.

An important contribution to making physically based rendering accessible and widely used was the hugely influential Physically Based Rendering book and PBRT renderer by Pharr and Humphreys [96]. (The first edition was printed in 2004, with updated and extended editions in 2010 and 2016.)

Now that production rendering is physically based, we can more easily transfer the academic research that has been done (and continues to be done) in physical light transport simulation to practical movie production.

6.2 Denoising

Traditionally, the Achilles' heel of path tracing has been noisy images and slow convergence (as shown in Figure 3.2). Despite importance

sampling, Russian roulette, and many other useful Monte Carlo noise reduction techniques, the error in each pixel is still proportional to the inverse square root of the number of samples: four times as many samples are needed to reduce the error by half. This convergence rate means that noise is reduced quickly initially, but then is reduced more and more slowly. Toward the end, waiting for an image to converge from "nearly good enough" to "good enough" can feel like watching paint dry (or, with a more technical term, has "diminishing returns"). This is where denoising comes in: it can eliminate that dreadful last wait.

A naive approach to denoising would be to blur the image uniformly, or to apply a standard photo denoising filter. But in rendered images, the amount of noise in the specular reflections can be very different from the noise in diffuse reflections (so they should be filtered over differing numbers of neighbor pixels), sharp texture details and object edges should be preserved, moving and out-of-focus objects need special treatment, etc. Fortunately the powerful and adaptive feature-guided denoisers that have been developed lately can remove objectionable noise from images without overblurring, eliminating the need to wait for path tracing to converge to a "noise-free" image. Figure 6.2 shows denoising on a frame from *Finding Dory*.

The denoiser is "feature-guided", meaning that for each pixel, it typically needs the average depth, surface normal, motion vector, surface albedo, specular and diffuse reflection, as well as variance estimates of each. This data can be written out by the renderer as separate image channels along with the main image. The denoiser is then run as a post-process. It can operate in two modes: either working on single frames in isolation, or on an entire sequence at a time (enabling the utilization of cross-frame image information). More details about modern denoising techniques can be found in the papers and course notes of Rouselle et al. [105], Zimmer et al. [144], and Zwicker et al. [145].

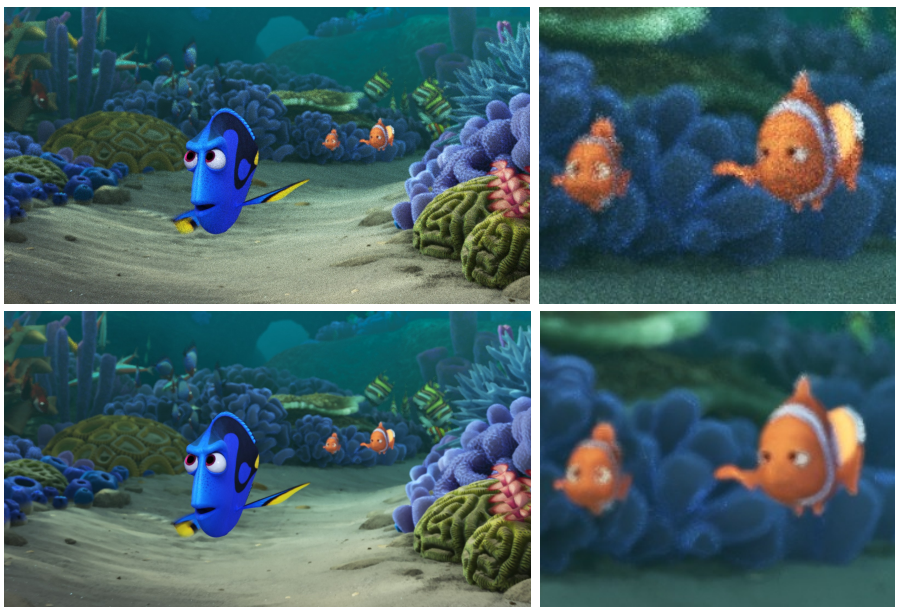

Figure 6.2: A frame from *Finding Dory* before (top) and after (bottom) denoising. The right column shows a close-up of both versions. (© 2016 Pixar/Disney.)

7

Why Path Tracing and Why Now?

In this section we discuss the main reasons why—in our view—path tracing has recently gained such popularity for movie rendering.

Path tracing is predictable, simple to learn, and simple to use under tight CG and VFX movie production deadlines. It has fewer "knobs" to tweak than traditional Reyes-based hybrid rendering methods. Path tracing is also very general, runs in a single pass, is simpler to multi-thread (under certain restrictions), and allows progressive rendering for quick feedback. The main disadvantages are noisy images, slow convergence, and large memory footprints. Most studios that have switched to path tracing have had to invest substantially in expanding their compute power, however, the reduction in man-hours outweighs the increased hardware costs.

A prerequisite for path tracing to become widely used in movie rendering was the development of efficient path-tracing methods to render realistic-looking surface materials, hair, fur, volumes, and subsurface scattering. (These are still areas of active research.)

Judicious use of efficient sampling techniques reduces image noise by orders of magnitude; the integration of these techniques in path-tracing renderers has been essential in providing the necessary performance

improvements that have made path tracing a viable rendering method for movies. Meanwhile, Moore's law has provided ever more powerful computers, with 4-, 8-, or 16-wide SIMD instructions and many-core processors. This has allowed us to generate images of higher visual complexity and resolution. The combination of better algorithms and more powerful computers has made path tracing a viable rendering solution, even with the extremely harsh "pixel perfect" requirements in the movie industry.

For many years path tracing was rejected outright due to the noisy images it produces. The noise is especially high in motion-blurred images, particularly around motion-blurred bright highlights. Some workarounds that have been used are to render without motion blur (not viable when rendering visual effects to be composited with real images), or to cut render times by only rendering every other or every third frame and then interpolate the in-between images (using motion vectors or optical flow). The advanced modern denoisers solve this problem.

Computers with larger memory allows us to render larger scenes, but scene complexity in movie production tends to grow even faster than the available memory. Path tracing inherits ray tracing's advantage of being able to use object instancing. Typical production scenes would not fit in memory without instancing.

Previously, studios would often use one renderer for initial scene layout and illumination setup, and another renderer for final frames. It is more convenient, however, to have the same renderer do both. With path tracing, the only difference between a quick preview image and a final movie frame is the level of noise and the time it takes to render.

Most rendering algorithms can be made progressive by first rendering low-quality images and then iteratively start over with higher quality settings. But with path tracing it is possible to simply continue the sample sequence in each pixel, thereby continually increasing the quality without having to start over. If left to render, a noisy preview image will progress to a low-noise final-quality image in a predictable manner.

A very practical and useful aspect of path tracing is the ability to write out (noisy) checkpoint images at regular intervals during ren-

dering, and later resume rendering from the last checkpoint. This fits nicely in the typical studio workflow where a sequence of images is rendered overnight for review the next morning. If these preliminary images are approved—i.e. all geometry, textures, lighting etc. are as they should be, but the images are still too noisy—then the rendering can be resumed from where it left off, simply adding new samples to each pixel.

8

Extensions and Challenges

The path tracing algorithm has been extended in various ways to deal with complex and challenging light paths. In this chapter we give an overview of these extensions and also mention some aspects that still need improvement.

8.1 Bidirectional path tracing

Bidirectional path tracing was developed independently by Lafortune and Willems [73], and Veach and Guibas [120]. The idea is to trace paths not only from the eye but also from the light sources, and then connect those paths. Figure 8.1 shows a subpath from the eye, a subpath from the light source, and all the potential full light transport paths formed by connecting the vertices across the two subpaths with shadow rays.

Bidirectional path tracing is advantageous for scenes dominated by indirect illumination or with strong caustics. Figure 8.2 shows a box illuminated by a small light source inside a wall sconce. Because the sconce blocks much of the direct light, most of the illumination in the scene is indirect. Unidirectional path tracing has a hard time finding

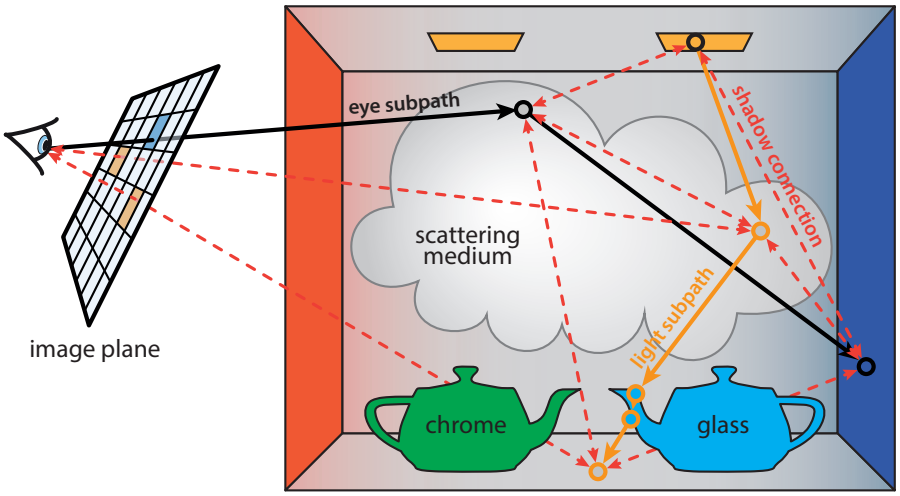

Figure 8.1: Bidirectional path tracing: paths from the eye and paths from light sources; potential connections shown with dashed red lines.

the light source, resulting in a lot of noise, whereas bidirectional path tracing explicitly traces paths from the light inside the sconce, and so handles this situation much better. The two images show that for this scene there is much lower noise with bidirectional path tracing than with unidirectional path tracing for equal rendering time.

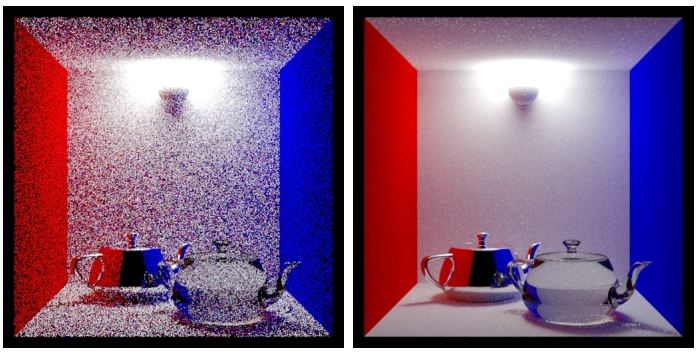

Figure 8.2: Unidirectional vs. bidirectional path tracing in a scene dominated by indirect illumination. Equal time renderers (2048 vs. 600 samples per pixel).

In practical use, there is an unexpected bonus from bidirectional path tracing: surface shader results (texture map lookups, procedural texture evaluations, etc.) are reused for several combined paths. This can give a nice speedup for the complex surface shaders typically employed in movie rendering.

A practical problem when bidirectional path tracing is used in complex production scenes is that incoherent texture access patterns for the light paths can cause texture cache thrashing. Ray differentials are not as useful as for unidirectional path tracing: just because a light path (photon) undergoes diffuse reflection (or specular reflection from a highly curved surface) unfortunately doesn't mean we can assign a a large differential to it since the light path could subsequently hit a surface point right in front of the eye. Even the recent work on covariance tracing [8] for bidirectional path tracing only specifies the optimal texture filter sizes after the light path has been traced and connected to the eye path; it does not solve the question of optimal filter sizes *during* light path tracing.

8.2 Metropolis

The Metropolis algorithm was introduced in a classic Monte Carlo paper [85]. When applied to rendering, Metropolis sampling excels at finding difficult paths from light sources to the eye, for example illumination from a neighboring room through a door that is only slighly ajar. Once a viable path has been found, mutations of it are explored to discover similar light transport paths. Veach and Guibas [122] introduced the Metropolis–Hastings variant [43] of the Metropolis algorithm to rendering, and Pauly et al. [91] extended it to volume rendering. Kelemen et al. [60] introduced the mutation strategy that is most popular for Metropolis rendering. Recent improvements in Metropolis include a gradient-domain approach [76, 79] that mostly follows image edges and reconstructs the final image using a Poisson solver. (Kettunen et al. [63] and Manzi et al. [80] showed that combining Monte Carlo pixel and gradient estimates using Poisson reconstruction can also be done outside the context of Metropolis using standard path tracing techniques.)

So far, Metropolis has not been used in movie production due to unpredictable pixel color "pops" as new important paths are discovered, making it hard to use for reliable interactive previews and difficult to guarantee frame-to-frame consistency in a movie sequence. But perhaps improved mutation strategies, such as those presented recently by Hachisuka et al. [41] and Li et al. [77], can overcome these practical limitations in the future.

8.3 Vertex connection and merging

Vertex Connection and Merging and Unified Path Sampling (which we will refer to as VCM/UPS) are effectively identical techniques independently developed by Georgiev et al. [33] and Hachisuka et al. [40], respectively. VCM/UPS combines bidirectional path tracing and progressive photon mapping, and is particularly advantageous for specular-diffuse paths and specular-diffuse-specular paths (i.e. caustics and specular reflections of caustics). Figure 8.3 shows a VCM/UPS rendering of a caustic under a glass teapot and a specular reflection of the caustic.

Figure 8.3: VCM/UPS rendering of a caustic and its reflection.

For progressive rendering, there is typically one eye path and one light path generated for each pixel in each iteration. These paths are connected, and at the same time the vertices along the light paths are stored in a photon map (which is used in the following iteration to avoid reading and writing in a single photon map at the same time).

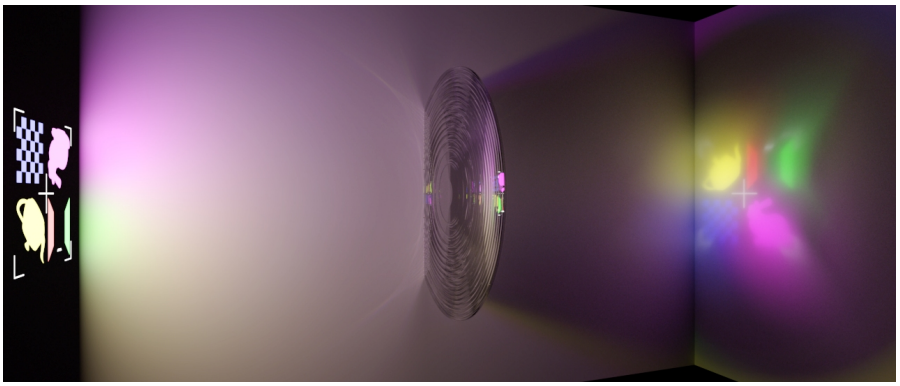

Figure 8.4: VCM/UPS rendering of refraction through a Fresnel lens. (Image courtesy of Andrew Kensler.)

Figure 8.4 shows another example of VCM/UPS rendering: a Fresnel lens refracting light from a textured light source (left) forming a projected image of the light source texture on the diffuse wall (right).

In practical use, VCM/UPS has the same problem as bidirectional path tracing in complex textured scenes: the light paths are incoherent, but it is hard to figure out path differentials that would safely allow coarse texture lookups without blurring the caustics. Another practical issue is that photon emission profiles and artistic light path manipulations must match the evaluations used for eye paths.

The VCM/UPS idea has been extended to volumes. Křivánek et al. [72] noted that sparse and dense volumes should be sampled differently to obtain least noise: point lookups [53] are best for dense volumes, while beam lookups [49, 50] are best for thin volumes. Their method uses both point and beam lookups, and combines the results with multiple importance sampling. They called the combined method Unified Points, Beams, and Paths (UPBP). UPBP can efficiently render all volume effects, but its particular strength is crepuscular rays, volume caustics, and specular reflections of volume caustics. Figure 8.5 shows two images rendered with UPBP: the first image is an illustration from the original research paper, the second image was rendered using RenderMan's UPBP implementation.

Figure 8.5: UPBP images of volumes with very different optical densities. (Luxo image: © 2016 Pixar.)

9

Discussion and Conclusion

Quite a few different methods have been used for rendering CG movies and visual effects over the past decades. The rendering community is now focusing on path tracing since it provides a unified framework and a simple single-pass workflow, it is suited both for quick previews and final-frame quality images, it can handle volumes, subsurface scattering, hair, fur, and other challenging cases, it scales reasonably well to large-scale multithreaded execution, and its Achilles' heel of noisy images and slow convergence has been mitigated with advanced denoising techniques.

More information about how to implement a path tracer (and didactic source code) can be found in the excellent PBRT book [96] or on the Mitsuba web page [48]. More details about path tracing in movie production can be found in the notes and slides for the recent SIGGRAPH course [61].

Several rendering teams have developed (or are in the process of developing) GPU-based path tracers. GPUs have immense computational power, but have lagged behind CPUs in the available memory and also require a different algorithm execution style than CPUs. With the huge computational complexity of movies, it will be interesting to

see which architecture wins. One possible outcome is that these architectures will merge over the next decade, in which case this becomes a moot point.

We look forward to seeing future improvements to path tracing, including solutions to the obstacles and challenges we have outlined in Chapter 8. One pleasant consequence of the movie industry moving to path tracing (and physically based rendering in general), is that the time gap between new academic research and use in the movie industry continues to shrink. Our hope is that this trend will continue to everyone's benefit, driving more rapid evolution of path tracing research with immediate practical usage.

Acknowledgements

Many thanks to Alexander Keller and Luca Fascione for organizing the 2015 SIGGRAPH course "The path tracing revolution in the movie industry" that inspired this survey, and to Brian Curless for suggesting that we should write it. Thanks to our current and former colleagues in Pixar's RenderMan team, Disney Research Zürich, and Dartmouth College for their support. Thanks to Brent Burley, Mark VandeWettering, Chris Kulla, Cliff Ramshaw, Christophe Hery, Philippe Leprince, Charlie Kilpatrick, David Laur and Wayne Wooten for detailed suggestions on how to improve the structure, contents, and readability of this paper. Also thanks to the image copyright holders for allowing us to use the movie images, and to the following people for helping us obtain images and the required permissions: Steve May, Tony Apodaca, Christophe Hery, Andrew Kensler, Stephen Friedman, Brent Burley, Dayna Meltzer, Philippe Leprince, Damien Fagnou, Eric Tabellion, Andrew Pearce, Chris Kulla, Erik Strauss, Luca Fascione, and Karl Ludwig.

References

[1] Attila Áfra, Carsten Benthin, Ingo Wald, and Jacob Munkberg. Local shading coherence extraction for SIMD-efficient path tracing on on CPUs. In *Proceedings of High Performance Graphics*, 2016.

[2] Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on GPUs. In *Proceedings of High Performance Graphics*, 2009.

[3] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *Proceedings of Eurographics*, pages 3–10, 1987.

[4] Anthony Apodaca and Larry Gritz. *Advanced RenderMan: Creating CGI for Motion Pictures*. Morgan Kaufmann, 2000.

[5] James Arvo and David Kirk. Particle transport and image synthesis. *Computer Graphics (Proceedings of SIGGRAPH)*, 24(4):63–66, 1990.

[6] Ronen Barzel. Lighting controls for computer cinematography. *Journal of Graphics Tools*, 2(1):1–20, 1997.

[7] Laurent Belcour, Cyril Soler, Kartic Subr, Nicholas Holzschuch, and Frédo Durand. 5D covariance tracing for efficient defocus and and motion blur. *ACM Transactions on Graphics*, 32(3):31, 2013.

[8] Laurent Belcour, Ling-Qi Yan, Ravi Ramamoorthi, and Derek Nowrouzezahrai. Antialiasing complex global illumination effects in path-space. Technical Report 1375, University of Montreal, 2015.

[9] Michael Bunnell. Dynamic ambient occlusion and indirect lighting. In Matt Pharr, editor, *GPU Gems 2*, pages 223–233. Addison-Wesley Publishers, 2005.

[10] Brent Burley. Extending the Disney BRDF to a BSDF with integrated subsurface scattering. In *'Physically Based Shading in Theory and Practice' SIGGRAPH Course*, 2015.

[11] Matt Jen-Yuan Chiang, Benedikt Bitterli, Chuck Tappan, and Brent Burley. A practical and controllable hair and fur model for production rendering. *Computer Graphics Forum (Proceedings of Eurographics)*, 35 (2):275–283, 2016.

[12] Matt Jen-Yuan Chiang, Peter Kutz, and Brent Burley. Practical and controllable subsurface scattering for production path tracing. In *SIGGRAPH Tech Talks*, 2016.

[13] Kenneth Chiu, Peter Shirley, and Changyaw Wang. Multi-jittered sampling. In *Graphics Gems IV*, chapter V.4, pages 370–374. Academic Press, 1994.

[14] Per Christensen. Point-based approximate color bleeding. Technical Report 08-01, Pixar Animation Studios, 2008.

[15] Per Christensen and Dana Batali. An irradiance atlas for global illumination in complex production scenes. *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*, pages 133–141, 2004.

[16] Per Christensen and Brent Burley. Approximate reflectance profiles for efficient subsurface scattering. Technical Report 15-04, Pixar Animation Studios, 2015.

[17] Per Christensen, David Laur, Julian Fong, Wayne Wooten, and Dana Batali. Ray differentials and multiresolution geometry caching for distribution ray tracing in complex scenes. *Computer Graphics Forum (Proceedings of Eurographics)*, 22(3):543–552, 2003.

[18] Per Christensen, Julian Fong, David Laur, and Dana Batali. Ray tracing for the movie 'Cars'. In *Proceedings of IEEE Symposium on Interactive Ray Tracing*, pages 1–6, 2006.

[19] Per Christensen, George Harker, Jonathan Shade, Brenden Schubert, and Dana Batali. Multiresolution radiosity caching for global illumination in movies. In *SIGGRAPH Tech Talks*, 2012.

[20] Andrew Clinton and Mark Elendt. Rendering volumes with microvoxels. In *SIGGRAPH Tech Talks*, 2009.

[21] Michael Cohen and John Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press, 1993.

[22] W. A. Coleman. Mathematical verification of a certain Monte Carlo sampling technique and applications of the technique to radiation transport problems. *Nuclear Science and Engineering*, 32(1):76–81, 1968.

[23] Robert Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, 1986.

[24] Robert Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics (Proceedings of SIGGRAPH)*, 18(3):137–145, 1984.

[25] Robert Cook, Loren Carpenter, and Edwin Catmull. The Reyes image rendering architecture. *Computer Graphics (Proceedings of SIGGRAPH)*, 21(4):95–102, 1987.

[26] R. Cranley and T. Patterson. Randomization of number theoretic methods for multiple integration. *SIAM Journal on Numerical Analysis*, 13: 904–914, 1976.

[27] Carsten Dachsbacher, Jaroslav Křivánek, Miloš Hašan, Adam Arbree, Bruce Walter, and Jan Novák. Scalable realistic rendering with many-light methods. *Computer Graphics Forum*, 33(1):88–104, 2014.

[28] Eugene d'Eon and Geoffrey Irving. A quantized-diffusion model for rendering translucent materials. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 30(4):56:1–56:14, 2011.

[29] Eugene d'Eon, Steven Marschner, and Johannes Hanika. Importance sampling for physically-based hair fiber models. In *SIGGRAPH Asia Technical Briefs*, 2013.

[30] Philip Dutré, Kavita Bala, and Philippe Bekaert. *Advanced Global Illumination*. A K Peters Ltd., second edition, 2005.

[31] Christian Eisenacher, Gregory Nichols, Andrew Selle, and Brent Burley. Sorted deferred shading for production path tracing. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 32(4):125–132, 2013.

[32] M. Galtier, S. Blanco, C. Caliot, C. Coustet, J. Dauchet, M. El Hafi, V. Eymet, R. Fournier, J. Gautrais, A. Khuong, B. Piaud, and G. Terrée. Integral formulation of null-collision Monte Carlo algorithms. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 125:57–68, 2013.

[33] Iliyan Georgiev, Jaroslav Křivánek, Tomas Davidovic, and Philipp Slusallek. Light transport simulation with vertex connection and merging. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 31(6), 2012.

[34] Iliyan Georgiev, Jaroslav Křivánek, Toshiya Hachisuka, Derek Nowrouzezahrai, and Wojciech Jarosz. Joint importance sampling of low-order volumetric scattering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 32(6), 2013.

[35] Andrew Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann, 1995.

[36] Leonhard Grünschloß. QMC sampling source code in C++, 2012. http://gruenschloss.org.

[37] Diego Gutierrez, Henrik Wann Jensen, Wojciech Jarosz, and Craig Donner. Scattering. In *SIGGRAPH Asia Courses*, 2009.

[38] Ralf Habel, Per Christensen, and Wojciech Jarosz. Photon beam diffusion: a hybrid Monte Carlo method for subsurface scattering. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 32(4):27–37, 2013.

[39] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 27(5), 2008.

[40] Toshiya Hachisuka, Jacopo Pantaleoni, and Henrik Wann Jensen. A path space extension for robust light transport simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 31(6), 2012.

[41] Toshiya Hachisuka, Anton Kaplanyan, and Carsten Dachsbacher. Multiplexed Metropolis light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 33(4), 2014.

[42] Johannes Hanika, Marc Droske, and Luca Fascione. Manifold next event estimation. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 34(4):87–97, 2015.

[43] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[44] Paul Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics (Proceedings of SIGGRAPH)*, 24(4):145–154, 1990.

[45] Christophe Hery and Ravi Ramamoorthi. Importance sampling of reflections from hair fibers. Technical Report 12-11, Pixar Animation Studios, 2012.

[46] Christophe Hery and Ryusuke Villemin. Physically based lighting at Pixar. In *'Physically Based Shading' SIGGRAPH Course*, 2013.

[47] Homan Igehy. Tracing ray differentials. *Proceedings of SIGGRAPH*, 33: 179–186, 1999.

[48] Wenzel Jakob. Mitsuba: Physically based renderer, 2010. http://www.mitsuba-renderer.org.

[49] Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. The beam radiance estimate for volumetric photon mapping. *Computer Graphics Forum (Proceedings of Eurographics)*, 27(2):557–566, 2008.

[50] Wojciech Jarosz, Derek Nowrouzezahrai, Iman Sadeghi, and Henrik Wann Jensen. A comprehensive theory of volumetric radiance estimation using photon points and beams. *ACM Transactions on Graphics*, 30(1):5:1–5:19, 2011.

[51] Henrik Wann Jensen. *Realistic Image Synthesis using Photon Mapping*. A K Peters Ltd., 2001.

[52] Henrik Wann Jensen. Global illumination using photon maps. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*, pages 21–30, 1996.

[53] Henrik Wann Jensen and Per Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. *Proceedings of SIGGRAPH*, 32:311–320, 1998.

[54] Henrik Wann Jensen, Steve Marschner, Marc Levoy, and Pat Hanrahan. A practical model for subsurface light transport. *Proceedings of SIGGRAPH*, 35:511–518, 2001.

[55] Henrik Wann Jensen, Per Christensen, Toshi Kato, and Frank Suykens. A practical guide to global illumination using photon mapping. In *SIGGRAPH Courses*, 2002.

[56] Stephen Joe and Frances Y. Kuo. Constructing Sobol' sequences with better two-dimensional projections. *SIAM Journal on Scientific Computation*, 30:2635–2654, 2008.

[57] Jim Kajiya. The rendering equation. *Computer Graphics (Proceedings of SIGGRAPH)*, 20(4):143–150, 1986.

[58] H. Kalli and E. Cashwell. Evaluation of three Monte Carlo estimation schemes for flux at a point. Technical Report LA-6865-MS, Los Alamos Scientific Laboratory, 1977.

[59] Malvin Kalos and Paula Whitlock. *Monte Carlo Methods*. John Wiley and Sons, 1986.

[60] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. A simple and robust mutation strategy for the Metropolis light transport algorithm. *Computer Graphics Forum (Proceedings of Eurographics)*, 21(3):531–540, 2002.

[61] Alexander Keller, Luca Fascione, Marcos Fajardo, Per Christensen, Johannes Hanika, Christian Eisenacher, and Greg Nichols. The path-tracing revolution in the movie industry. In *SIGGRAPH Courses*, 2015.

[62] Andrew Kensler. Correlated multi-jittered sampling. Technical Report TM-13-01, Pixar Animation Studios, 2013.

[63] Markus Kettunen, Marco Manzi, Miika Aittala, Jaakko Lehtinen, Frédo Durand, and Matthias Zwicker. Gradient-domain path tracing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 34(4), 2015.

[64] Alan King, Christopher Kulla, Alejandro Conty, and Marcos Fajardo. BSSRDF importance sampling. In *SIGGRAPH Tech Talks*, 2013.

[65] Claude Knaus and Matthias Zwicker. Progressive photon mapping: a probabilistic approach. *ACM Transactions on Graphics*, 30(3), 2011.

[66] Donald E. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, 3rd edition, 1998.

[67] Craig Kolb, Don Mitchell, and Pat Hanrahan. A realistic camera model for computer graphics. *Proceedings of SIGGRAPH*, 29:317–324, 1995.

[68] Thomas Kollig and Alexander Keller. Efficient multidimensional sampling. *Computer Graphics Forum (Proceedings of Eurographics)*, 21(3): 557–563, 2002.

[69] Janne Kontkanen, Eric Tabellion, and Ryan Overbeck. Coherent out-of-core point-based global illumination. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 30(4):1353–1360, 2011.

[70] Christopher Kulla and Marcos Fajardo. Importance sampling techniques for path tracing in participating media. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 31 (4):1519–1528, 2012.

[71] Jaroslav Křivánek, Pascal Gautron, Greg Ward, Henrik Wann Jensen, Eric Tabellion, and Per Christensen. Practical global illumination with irradiance caching. In *SIGGRAPH Courses*, 2008.

[72] Jaroslav Křivánek, Iliyan Georgiev, Toshiya Hachisuka, Petr Vévoda, Martin Šik, Derek Nowrouzezahrai, and Wojciech Jarosz. Unifying points, beams, and paths in volumetric light transport simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 33(4), 2014.

[73] Eric Lafortune and Yves Willems. Bi-directional path tracing. In *Proceedings of Compugraphics*, pages 145–153, 1993.

[74] Eric Lafortune and Yves Willems. Rendering participating media with bidirectional path tracing. *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*, pages 91–101, 1996.

[75] Samuli Laine, Tero Karras, and Timo Aila. Megakernels considered harmful: wavefront path tracing on GPUs. In *Proceedings of High Performance Graphics*, 2013.

[76] Jaakko Lehtinen, Tero Karras, Samuli Laine, Miika Aittala, Frédo Durand, and Timo Aila. Gradient-domain Metropolis light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 32(4), 2013.

[77] Tzu-Mao Li, Jaakko Lehtinen, Ravi Ramamoorthi, Wenzel Jacob, and Frédo Durand. Anisotropic Gaussian mutations for Metropolis light transport through Hessian-Hamiltonian dynamics. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 34(6), 2015.

[78] Tom Lokovic and Eric Veach. Deep shadow maps. In *Proceedings of SIGGRAPH*, pages 385–392, 2000.

[79] Marco Manzi, Fabrice Rousselle, Markus Kettunen, Jaakko Lehtinen, and Matthias Zwicker. Improved sampling for gradient-domain Metropolis light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 33(6), 2014.

[80] Marco Manzi, Markus Kettunen, Miika Aittala, Jaakko Lehtinen, Frédo Durand, and Matthias Zwicker. Gradient-domain bidirectional path tracing. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 34, 2015.

[81] Stephen Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan. Light scattering from human hair fibers. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 22(3): 780–791, 2003.

[82] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1): 3–30, 1998.

[83] Johannes Meng, Johannes Hanika, and Carsten Dachsbacher. Improving the Dwivedi sampling scheme. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 35(4):37–44, 2016.

[84] Nick Metropolis. The beginning of the Monte Carlo method. *Los Alamos Science (special issue)*, pages 125–130, 1987.

[85] Nick Metropolis, Arianna Rosenbluth, Marshall Rosenbluth, Augusta Teller, and Edward Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[86] Don Mitchell. Spectrally optimal sampling for distribution ray tracing. *Computer Graphics (Proceedings of SIGGRAPH)*, 25(4):157–164, 1991.

[87] Jan Novák, Andrew Selle, and Wojciech Jarosz. Residual ratio tracking for estimating attenuation in participating media. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 33(6), 2014.

[88] Derek Nowrouzezahrai, Jared Johnson, Andrew Selle, Dylan Lacewell, Michael Kaschalk, and Wojciech Jarosz. A programmable system for artistic volumetric lighting. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 30(4), 2011.

[89] Art Owen. Monte Carlo variance of scrambled net quadrature. *SIAM Journal on Numerical Analysis*, 34:1884–1910, 1997.

[90] Steven Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. OptiX: A general purpose ray tracing engine. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 29(4), 2010.

[91] Mark Pauly, Thomas Kollig, and Alexander Keller. Metropolis light transport for participating media. *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*, pages 11–22, 2000.

[92] Leonid Pekelis, Christophe Hery, Ryusuke Villemin, and Junyi Ling. A data-driven light scattering model for hair. Technical Report 15-02, Pixar Animation Studios, 2015.

[93] Fabio Pellacini, Kiril Vidimče, Aaron Lefohn, Alex Mohr, Mark Leone, and John Warren. Lpics: a hybrid hardware-accelerated relighting engine for computer cinematography. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 24(3):464–470, 2005.

[94] Ken Perlin and Eric Hoffert. Hypertexture. *Computer Graphics (Proceedings of SIGGRAPH)*, 23(3):253–262, 1989.

[95] Matt Pharr and Pat Hanrahan. Geometry caching for ray-tracing displacement maps. *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*, pages 31–40, 1996.

[96] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2nd edition, 2010.

[97] Matt Pharr, Craig Kolb, Reid Gerschbein, and Pat Hanrahan. Rendering complex scenes with memory-coherent ray tracing. *Proceedings of SIGGRAPH*, 31:101–108, 1997.

[98] Matthias Raab, Daniel Seibert, and Alexander Keller. Unbiased global illumination with participating media. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 591–606. Springer, 2008.

[99] Michal Radziszewski, Krzysztof Boryczko, and Witold Alda. An improved technique for full spectral rendering. *Journal of WSCG*, 17(1-3): 9–16, 2009.

[100] Jonathan Ragan-Kelley, Charlie Kilpatrick, Brian Smith, Doug Epps, Paul Green, Christophe Hery, and Frédo Durand. The Lightspeed automatic interactive lighting preview system. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 26(3), 2007.

[101] Ravi Ramamoorthi, John Anderson, Mark Meyer, and Derek Nowrouzezahrai. A theory of Monte Carlo visibility sampling. *ACM Transactions on Graphics*, 31(5), 2012.

[102] Bernhard Reinert, Tobias Ritschel, Hans-Peter Seidel, and Iliyan Georgiev. Projective blue-noise sampling. *Computer Graphics Forum*, 35:285–295, 2015.

[103] Herbert Rief, A. Dubi, and Tov Elperin. Track length estimation applied to point detector. *Nuclear Science and Engineering*, 87:59–71, 1984.

[104] Tobias Ritschel, Thomas Engelhardt, Thorsten Grosch, Hans-Peter Seidel, Jan Kautz, and Carsten Dachsbacher. Micro-rendering for scalable, parallel final gathering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 28(5), 2009.

[105] Fabrice Rouselle, Marco Manzi, and Matthias Zwicker. Robust denoising using feature and color information. *Computer Graphics Forum (Proceedings of Pacific Graphics)*, 32(7):121–130, 2013.

[106] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-D shapes. *Computer Graphics (Proceedings of SIGGRAPH)*, 24(4): 199–206, 1990.

[107] Thorsten-Walther Schmidt, Jan Novák, Johannes Meng, Anton Kaplanyan, Tim Reiner, Derek Nowrouzezahrai, and Carsten Dachsbacher. Path-space manipulation of physically-based light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 32(4), 2013.

[108] Thorsten-Walther Schmidt, Fabio Pellacini, Derek Nowrouzezahrai, Wojciech Jarosz, and Carsten Dachsbacher. State of the art in artistic editing of appearance, lighting, and material. *Computer Graphics Forum*, 35(1):216–233, 2016.

[109] Jorge Schwarzhaupt, Henrik Wann Jensen, and Wojciech Jarosz. Practical Hessian-based error control for irradiance caching. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 31(6), 2012.

[110] Peter Shirley. *Realistic Ray Tracing*. A K Peters Ltd., 2000.

[111] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte Carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36, 1996.

[112] Helge Skullerud. The stochastic computer simulation of ion motion in a gas subjected to a constant electric field. *Journal of Physics D: Applied Physics*, 1(11):1567–1568, 1968.

[113] Ben Snow. Terminators and Iron Men. In *'Physically Based Shading Models in Film and Game Production' SIGGRAPH Course*, 2010.

[114] Jerome Spanier and Ely Gelbard. *Monte Carlo Principles and Neutron Transport Problems*. Addison-Wesley, 1969.

[115] Kartic Subr, Derek Nowrouzezahrai, Wojciech Jarosz, Jan Kautz, and Kenny Mitchell. Error analysis of estimators that use combinations of stochastic sampling strategies for direct illumination. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 33(4):93–102, 2014.

[116] Frank Suykens and Yves Willems. Path differentials and applications. *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*, pages 257–268, 2001.

[117] László Szirmay-Kalos, Balázs Tóth, and Milán Magdics. Free path sampling in high resolution inhomogeneous participating media. *Computer Graphics Forum*, 30(1):85–97, 2011.

[118] Eric Tabellion and Arnauld Lamorlette. An approximate global illumination system for computer generated films. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 23(3):469–476, 2004.

[119] Yusuke Tokuyoshi and Takahiro Harada. Stochastic light culling. *Journal of Computer Graphics Techniques*, 5(1):35–60, 2016.

[120] Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. In *Proceedings of the Eurographics Workshop on Rendering*, pages 147–162, 1994.

[121] Eric Veach and Leonidas Guibas. Optimally combining sampling techniques for Monte Carlo rendering. *Proceedings of SIGGRAPH*, 29:419–428, 1995.

[122] Eric Veach and Leonidas Guibas. Metropolis light transport. *Proceedings of SIGGRAPH*, 31:65–76, 1997.

[123] Ryusuke Villemin and Christophe Hery. Practical illumination from flames. *Journal of Computer Graphics Techniques*, 2(2):142–155, 2013.

[124] Jiři Vorba, Ondřej Karlik, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. On-line learning of parameteric mixture models for light transport simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 33(4), 2014.

[125] Ingo Wald, Philipp Slusallek, and Carsten Benthin. Interactive distributed ray tracing of highly complex models. *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*, pages 274–285, 2001.

[126] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum (Proceedings of Eurographics)*, 20(3):153–164, 2001.

[127] Ingo Wald, Sven Woop, Carsten Benthin, Gregory Johnson, and Manfred Ernst. Embree: a kernel framework for efficient CPU ray tracing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 33(4), 2014.

[128] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald Greenberg. Lightcuts: a scalable approach to illumination. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 24(3):1098–1107, 2005.

[129] Bruce Walter, Adam Arbree, Kavita Bala, and Donald Greenberg. Multidimensional lightcuts. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 25(3):1081–1088, 2006.

[130] Bruce Walter, Pramook Khungurn, and Kavita Bala. Bidirectional lightcuts. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 31 (4):59:1–59:11, 2012.

[131] Gregory Ward. Adaptive shadow testing for ray tracing. In *Proceedings of the Eurographics Workshop on Rendering*, pages 11–20, 1991.

[132] Gregory Ward and Paul Heckbert. Irradiance gradients. In *Proceedings of the Eurographics Workshop on Rendering*, pages 85–98, 1992.

[133] Gregory Ward, Francis Rubinstein, and Robert Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics (Proceedings of SIGGRAPH)*, 22(4):85–92, 1988.

[134] Greg Ward Larson and Rob Shakespeare. *Rendering with Radiance: The Art and Science of Lighting Visualization*. Morgan Kaufmann, 1998.

[135] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, 1980.

[136] Alexander Wilkie, Sehera Nawaz, Marc Droske, Andrea Weidlich, and Johannes Hanika. Hero wavelength spectral sampling. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 33(4):123–131, 2014.

[137] Lance Williams. Pyramidal parametrics. *Computer Graphics (Proceedings of SIGGRAPH)*, 17(3):1–11, 1983.

[138] E.R. Woodcock, T. Murphy, P. Hemmings, and T. Longworth. Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. In *Applications of Computing Methods to Reactor Problems*. Argonne National Laboratory, 1965.

[139] Sven Woop, Carsten Benthin, Ingo Wald, Gregory Johnson, and Eric Tabellion. Exploiting local orientation similarity for efficient ray traversal of hair and fur. In *Proceedings of High Performance Graphics*, 2014.

[140] Magnus Wrenninge. *Production Volume Rendering: Design and Implementation*. A K Peters Ltd/CRC Press, 2012.

[141] Magnus Wrenninge. Efficient rendering of volumetric motion blur using temporally unstructured volumes. *Journal of Computer Graphics Techniques*, 5(1), 2016.

[142] Ling-Qi Yan, Chi-Wei Tseng, Henrik Wann Jensen, and Ravi Ramamoorthi. Physically-accurate fur reflectance: modelling, measurement and rendering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 34(6), 2015.

[143] Yonghao Yue, Kei Iwasaki, Bing-Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. Unbiased, adaptive stochastic sampling for rendering inhomogeneous participating media. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 29(6), 2010.

[144] Henning Zimmer, Fabrice Rouselle, Wenzel Jakob, Oliver Wang, David Adler, Wojciech Jarosz, Olga Sorkine-Hornung, and Alexander Sorkine-Hornung. Path-space motion estimation and decomposition for robust animation filtering. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 34(4):131–142, 2015.

[145] Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rouselle, Pradeep Sen, Cyril Soler, and Sung-Eui Yoon. Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. In *Eurographics STAR Reports*, 2015.