Guest Editor's Introduction: Special Issue on Production Rendering

MATT PHARR, Google, Inc.

CCS Concepts: • **Computing methodologies** → **Rendering**; **Ray tracing**; *Computer graphics*;

Additional Key Words and Phrases: Path tracing, production rendering, graphics software systems

ACM Reference format:
Matt Pharr. 2018. Guest Editor's Introduction: Special Issue on Production Rendering. *ACM Trans. Graph.* 37, 3, Article 28 (July 2018), 4 pages.
https://doi.org/10.1145/3212511

1 HISTORY

3D computer graphics has revolutionized filmmaking. Computer generated imagery (CGI) has become the foundation of modern visual effects. It allows virtual characters, physical phenomena, and environments to be seamlessly integrated with footage of the real world; moviegoers regularly see cities being destroyed by monsters and lush landscapes of alien worlds, all of which are simulated on computers and rendered as images that are visually indistinguishable from reality. Starting with *Toy Story*, CGI has also made a new form of animated feature film possible, where expressive 3D computer-generated characters perform in rich virtual environments.

The capabilities of CGI have progressed rapidly over the nearly three decades that it has been widely used in movies. If one compares *Jurassic Park* and *Toy Story*, groundbreaking early visual effects and CGI animated movies, respectively, to modern counterparts like *War for the Planet of the Apes* and *Coco*, the advancement has been breathtaking: environments are much more complex, the lighting is realistic, and large-scale physical simulation has made high-fidelity flowing water, smoke, and fire possible.

Many factors have contributed to these improvements, including more powerful processors, better interactive tools for artists, and advances in the mathematics of physical simulation and animation, but advances in rendering have been critical. Rendering computes the final pixel colors; a renderer must not only do no harm to the scenes given to it, but it must deliver results that are acceptable to the human visual system, which is highly attuned to detecting when something is not quite right.

From the late 1980s until a few years after 2000, rendering for film was largely a monoculture: a single algorithm, Reyes (Cook et al. 1987), and a single implementation, Pixar's Photorealistic RenderMan, were used ubiquitously.

The Reyes algorithm was designed from the start to address the demands of film production. Its foundation is a high-quality rasterization algorithm that can efficiently render anti-aliased images with motion blur and depth of field, and computation is structured such that only a small subset of the scene's geometric and texture data must be in memory at any time, allowing Reyes to render complex scenes on systems with limited main memory. Cook et al.'s paper on Reyes is a classic; it's one where no matter how many times one has read it, one always learns something new.

The last 10 or so years have seen a remarkably fast shift away from Reyes. Today, not only are a multitude of different renderers now used for film production, but they're based on a completely different algorithm—path tracing. Introduced to computer graphics by Jim Kajiya (1986), path tracing uses Monte Carlo integration to model global illumination: the effect of light scattering multiple times as it illuminates surfaces.[1]

Unlike previous global illumination algorithms, path tracing places few limits on geometric representation, surface reflection models, or the types of light sources that can be supported. However, path tracing long seemed to be utterly unsuitable for feature films; see Figure 1, which compares an image from Kajiya's 1986 paper and an image from the 1987 paper on Reyes. Kajiya's image took about the same amount of time to render, though at a much lower resolution. Although the subtle lighting effects are impressive (and had never been seen at the time), his scenes themselves were simple.

It wasn't just computational demands that made path tracing unsuitable for production rendering: path tracing also expects that the entire scene description will be available in main memory. In mid-1985, a gigabyte of RAM cost approximately $500,000 (McCallum 2017); thus, only a few megabytes of RAM were available on the machines used for production rendering at the time, making it impossible to render complex scenes with path tracing. Anyone serious about making feature films naturally chose what Reyes made available to them on the computers of the day—geometric and texture complexity—over the lighting complexity that path tracing offered.[2]

As RAM became less expensive and computers came to have more and more memory, Reyes delivered imagery with increasing amounts of geometric and texture complexity. (Consider, for example, the lush world of ant-sized plants in *A Bug's Life*.) However, richer and more accurate lighting became increasingly desirable for feature films as improving the realism of illumination became the most effective way to improve image realism. A variety of efforts were undertaken to build hybrid rendering systems

Author's address: M. Pharr, Google, Inc. 1600 Amphitheatre Parkway, Mountain View, CA, 94043; email: matt@pharr.org.

0730-0301/2018/07-ART28
https://doi.org/10.1145/3212511

[1] For a gentle introduction to path tracing, see Pete Shirley's *Ray Tracing in One Weekend* book (Shirley 2016).
[2] With one important exception: Blue Sky Studios pursued a ray-tracing based global illumination approach beginning in the late 1980s. In 1998, their Oscar-winning short, *Bunny*, caught the attention of many with the beautiful soft lighting that their global illumination algorithms made possible. Unfortunately, Blue Sky has published almost no technical information about the implementation of their renderer, CGI Studio.

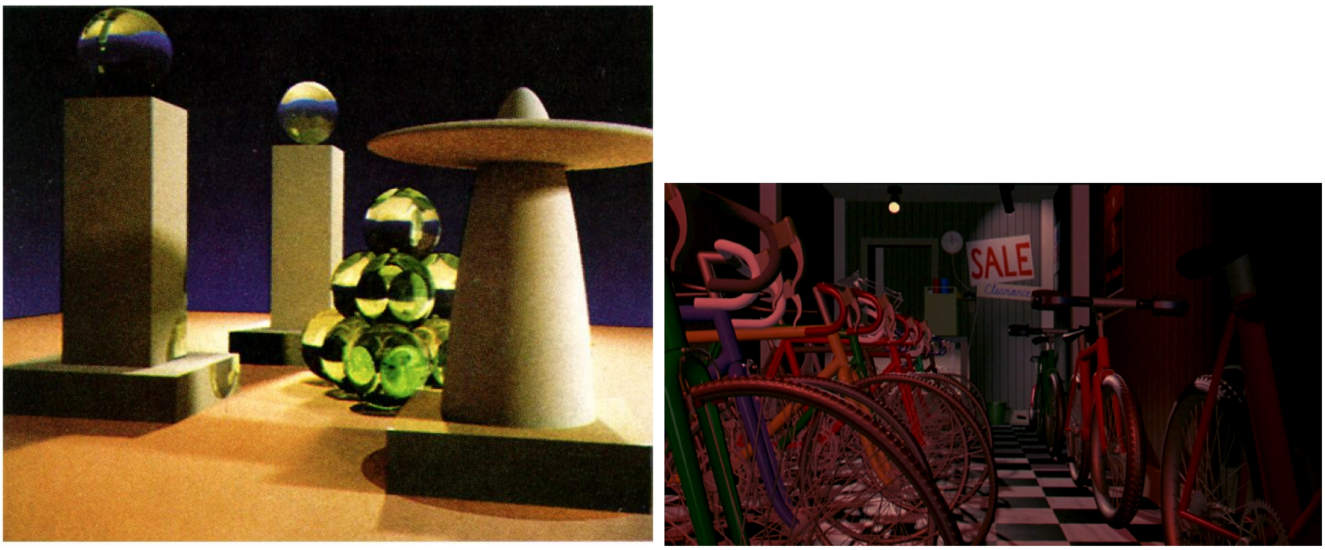

Fig. 1. Left, an image generated with path tracing by Jim Kajiya in 1985. Though the scene is simple, it still required 7 hours to render at 256x256 resolution. However, the lighting effects—soft shadows, indirect illumination, and caustics—had never before been seen. Right, an image rendered with Reyes in 1987. This scene was highly complex for the era and was slightly faster to render than Kajiya's scene at 6 hours 27 minutes, but it was rendered at 1024x768 resolution and with much more geometric and texture detail on a system with 32MB of RAM (Reeves and Ostby 2018). However, all non-local lighting effects like shadows and reflections had to be approximated with shadow and reflection maps.

with Reyes as a foundation and either selectively used ray tracing (Apodaca and Gritz 1999; Christensen et al. 2006; Tabellion and Lamorlette 2004) or used various precomputed global illumination lighting representations (Christensen and Batali 2004; Pantaleoni et al. 2010).

While these approaches succeeded in bringing ray tracing and global illumination effects to Reyes, they led to unwieldy workflows. The discussions in the papers by Christensen et al. and Fascione et al. in this issue describe the challenges that these approaches brought with them in detail. When these approaches were in use, most practitioners would have agreed that given unlimited computing resources, path tracing would be the preferred approach for its generality, but also would have agreed that, as a practical matter, it was still too expensive.

Starting in the early 2000s, a number of developments outside of film production made it reasonable to begin to consider writing production rendering systems that were based on path tracing:

—Many advancements were made in Monte Carlo techniques for rendering, including more effective light sampling algorithms (Shirley et al. 1996), high-quality sampling patterns (Kollig and Keller 2002), and multiple importance sampling (Veach and Guibas 1995), which together greatly reduced the variance of path tracing, giving high-quality images with fewer rays traced, and thus, less computational cost.
—Gigabytes of RAM became affordable—in 2005, 1GB of DRAM cost less than $200. Thus, it started to become possible to store the entirety of complex scenes in main memory.
—Progress in high-performance ray tracing and better techniques for building acceleration structures made implementations of the core ray tracing algorithm that path tracing uses much more efficient (Wald et al. 2001).
—CPUs continued to become faster (and more parallel); multiple CPU cores became available on systems used for rendering, making the computational demands of path tracing more acceptable. Further, path tracing is more amenable to scalable parallel implementation than Reyes, so it could make better use of multi-core CPUs.

While these developments made it possible to imagine using path tracing for production rendering, a number of problems specific to rendering for film still had to be solved. Texture complexity was one of the most important ones: loading all of the texture maps of a production scene into RAM remained infeasible and the ability to maintain a texture cache that only kept a small fraction of the scene's texture in main memory was a necessity. Developers found that the combination of texture caching (Peachey 1990) and ray differentials (Christensen et al. 2003; Gritz and Hahn 1996; Igehy 1999) to extend Peachey's "principle of texture thrift" to path tracing made it possible to render scenes with small texture caches of just a few gigabytes of RAM.

Computing accurate global illumination also required finding ways to incorporate specialized effects into the path tracing algorithm. Previously, effects like volumetric scattering (e.g., clouds, smoke, and explosions), hair and fur, and subsurface scattering were often handled by different renderers and required specialized techniques for effects like shadows and reflections. Including these in path tracing's light scattering simulation required a combination of applying techniques from the research literature and developing new ones; many of the details are described in the articles in this issue. For an in-depth look at this transition, see the work of Christensen and Jarosz (2016).

As these problems were solved and path tracing started to see adoption in studios, users soon found that in addition to

eliminating the complex Reyes-based pipelines, path tracing offered useful additional advantages over Reyes, including progressive refinement of low-quality images, the ability to easily stop, checkpoint, and restart rendering. There was also the surprising fact that noisy images that have been rendered with just a few samples per pixel could still be useful for animation reviews and finalizing shading and lighting in shots; multiple studios have reported taking advantage of this (Fascione et al. 2017). Thus, the expensive final rendering of a frame might only need to happen once during production, rather than multiple times as it made its way through the production pipeline as was done with Reyes.

Just as remarkable as the shift to path tracing is the fact that many studios are now writing their own custom renderers. For a studio, the tradeoff is not an easy one: more control and customizability and the ability to fix bugs quickly with an in-house renderer versus the fact that the commercial systems have been used on hundreds of movies and are thus likely to be more robust than in-house systems that have been used on fewer. However, the very fact that this choice can be considered is a significant shift from the days of Reyes.

The availability of a wide variety of open source software libraries has made it easier to write an in-house renderer. With libraries like Embree, OpenSubdiv, Open Shading Language, Open ImageIO, OpenColorIO, OpenVDB, Ptex, OpenEXR, and Alembic available with permissive licenses, many of the key ingredients needed to develop a production renderer are freely available. The open source versions of these are one and the same as the versions used throughout film production pipelines at most studios, which ensures that they are robust and well-tested.

Finally, widespread availability of information about path tracing, Monte Carlo rendering, and physically based rendering has also helped facilitate this transition. Most research papers are easily found online, there are now entire books on path tracing and physically based rendering available (Pharr et al. 2016). If one is trying to implement a tricky rendering algorithm, one can often find blog posts where others have written about how they've implemented it and there are a multitude of open source path tracers that also serve as a useful reference about implementation details.

2 ISSUE OVERVIEW

We were fortunate that the developers of most of the major rendering systems used in film production have been willing to write articles about their systems for this issue. Almost all published rendering research is algorithm-focused, but a successful production system is much more than a bunch of algorithms strung together. These systems must be highly robust (both in their execution and in the results they generate), interoperate with complex production pipelines and exhibit fairly predictable behavior in output, runtime, and memory use. The constraints of production rendering and the algorithms that have been developed to address these constraints are rarely discussed in the research literature.

This issue includes articles on five renderers:

- Arnold (Solid Angle), described by Georgiev et al.: a commercial renderer used on hundreds of feature films and numerous commercials and TV series since 2001.
- Arnold (Sony Pictures Imageworks), described by Kulla et al.: Sony's proprietary version of Arnold, used on all of Sony's movies since 2008 after early success on the film *Monster House* (2006).
- Hyperion (Walt Disney Animation Studios), described by Burley et al.: used on all of the studio's animated movies starting with *Big Hero 6* (2014).
- Manuka (Weta Digital), described by Fascione et al.: used on all of Weta's films starting with *Dawn of the Planet of the Apes* (2014).
- RenderMan[3] (Pixar Animation Studios), described by Christensen et al.: both a commercial product, used on over 45 feature films, and the renderer used by Pixar for all of its movies since *Finding Dory* (2016).

These renderers are built on a few common foundations: all use path tracing and employ bounding volume hierarchies for ray intersection acceleration. All support a range of geometric primitives, including subdivision surfaces, hair, and scattering volumes, and all run on CPUs and are multi-threaded, with varying levels of adoption of single instruction, multiple data (SIMD) processing. The challenges of graphics processing unit (GPU) rendering (primarily, limited local memory on GPUs) have so far precluded any of these systems also using GPUs, though work in this area continues, given the computational capabilities they offer.

Beyond that core, there's a remarkable amount of variety among them; different system designers have made different implementation decisions in a number of fundamental areas. Examples include:

- The variety of light transport algorithms supported, ranging from just path tracing (the Solid Angle version of Arnold), to path tracing with selective photon mapping (Hyperion), to a wide variety of global illumination algorithms (Manuka, RenderMan, Sony's Arnold).
- When texture mapping and shading occurs: in Manuka, all of it is done before rendering begins, allowing efficient BRDF generation at intersection points during rendering, with no need for a texture cache at that point; other systems do this the conventional way—at ray intersection time.
- Whether the renderer is sold commercially (Solid Angle Arnold and RenderMan): commercial products must serve a wider variety of workflows than in-house renderers and must include features that let the end-user customize their behavior. For an in-house renderer, therefore, simpler software architectures can be possible.
- Specialization to studio workflows and needs: Hyperion's design was strongly influenced by the need for artistic control, while Manuka places particular emphasis on grounding rendering in measured data for predictably matching on-set measurements.
- Whether "out of core" rendering is supported: Hyperion batches and sorts rays in order to access geometry and texture more coherently while other systems do not, accepting incoherent memory access from path tracing.

[3] An all new path-tracer that has kept Pixar's Reyes renderer's name.

The detailed analyses of these systems in these articles is a great gift to the broader graphics research community. Not only are there many new ideas and insights throughout all of these articles, but they provide a window into the biggest challenges that still face production rendering, which should be of benefit to rendering researchers.

ACKNOWLEDGMENTS

The idea for this special issue of *Transactions on Graphics* is due to Luca Fascione, who had the insight that the best way to convince the developers of production rendering systems to write about them in depth would be for everyone to do it simultaneously. This issue wouldn't have been possible without Kavita Bala's support of this project as Editor-in-Chief of *ACM Transactions on Graphics*.

Thanks also to Pete Shirley and Kayvon Fatahalian for help with reviewing the submissions and providing feedback to the authors. Finally, many thanks to the authors of these articles—both for their willingness to share many new details of their systems and the insights behind them as well as their good cheer in going through the multiple editing cycles while being busy with their day jobs.

REFERENCES

Tony A. Apodaca and Larry Gritz. 1999. *Advanced Renderman: Creating CGI for Motion Pictures*. Morgan Kaufmann, San Francisco, CA.

Per H. Christensen and Dana Batali. 2004. An irradiance atlas for global illumination in complex production scenes. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques (EGSR'04)*. 133–141. DOI: http://dx.doi.org/10.2312/EGWR/EGSR04/133-141

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

Per H. Christensen and Wojciech Jarosz. 2016. The path to path-traced movies. *Foundations and Trends in Computer Graphics and Vision* 10, 2 (2016), 103–175.

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

Robert L. Cook, Loren Carpenter, and Edwin Catmull. 1987. The Reyes image rendering architecture. *Computer Graphics (Proceedings of SIGGRAPH)* (Aug. 1987), 95–102. DOI: http://dx.doi.org/10.1145/37402.37414

Luca Fascione, Johannes Hanika, Marcos Fajardo, Per Christensen, Brent Burley, and Brian Green. 2017. Path tracing in production—Part 1: Production renderers. *ACM SIGGRAPH 2017 Courses* (2017). DOI: http://dx.doi.org/10.1145/3084873.3084904

Larry Gritz and James K. Hahn. 1996. BMRT: A global illumination implementation of the RenderMan standard. *Journal of Graphics Tools* 1, 3 (1996), 29–47.

Homan Igehy. 1999. Tracing ray differentials. *Computer Graphics (Proceedings of SIGGRAPH)* (1999), 179–186. DOI: http://dx.doi.org/10.1145/311535.311555

James T. Kajiya. 1986. The rendering equation. *Computer Graphics (Proceedings of SIGGRAPH)* (1986), 143–150. DOI: http://dx.doi.org/10.1145/15922.15902

Thomas Kollig and Alexander Keller. 2002. Efficient multidimensional sampling. *Computer Graphics Forum (Proceedings of Eurographcis)* 21 (2002), 557–563. DOI: http://dx.doi.org/10.1111/1467-8659.00706

John C. McCallum. 2017. Memory Prices (1957–2017). Retrieved from https://jcmit.net/memoryprice.htm.

Jacopo Pantaleoni, Luca Fascione, Martin Hill, and Timo Aila. 2010. PantaRay: Fast ray-traced occlusion caching of massive scenes. *ACM SIGGRAPH 2010 papers* (2010). DOI: http://dx.doi.org/10.1145/1833349.1778774

Darwyn Peachey. 1990. Texture on Demand. (1990). Animation Studios Technical Memo #217.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering: From Theory to Implementation*. Elsevier.

Bill Reeves and Eben Ostby. 2018. *Personal communication*. (2018).

Peter Shirley. 2016. *Ray Tracing in One Weekend*.

Peter Shirley, Changyaw Wang, and Kurt Zimmerman. 1996. Monte Carlo techniques for direct lighting calculations. *ACM Transactions on Graphics* 15, 1 (1996), 1–36. DOI: http://dx.doi.org/10.1145/226150.226151

Eric Tabellion and Arnauld Lamorlette. 2004. An approximate global illumination system for computer generated films. *Computer Graphics (Proceedings of SIGGRAPH)* (2004), 469–476. DOI: http://dx.doi.org/10.1145/1186562.1015748

Eric Veach and Leonidas J. Guibas. 1995. Optimally combining sampling techniques for Monte Carlo rendering. *Computer Graphics (Proceedings of SIGGRAPH)* (1995), 419–428. DOI: http://dx.doi.org/10.1145/218380.218498

Ingo Wald, Phillip Slusallek, Carsten Benthin, and Markus Wagner. 2001. Interactive rendering with coherent ray tracing. *Computer Graphics Forum* 20 (2001), 153–165. DOI: http://dx.doi.org/10.1111/1467-8659.00508