Manuka: A batch-shading architecture for spectral path tracing in movie production

LUCA FASCIONE, JOHANNES HANIKA, MARK LEONE, MARC DROSKE, JORGE SCHWARZHAUPT, TOMÁŠ DAVIDOVIČ, ANDREA WEIDLICH, and JOHANNES MENG, Weta Digital

Fig. 1: A frame from the *War for the Planet of the Apes* movie, rendered in Manuka.
Image courtesy of Weta Digital, ©2017 Twentieth Century Fox Film Corporation. All rights reserved.

The *Manuka* rendering architecture has been designed in the spirit of the classic REYES rendering architecture: to enable the creation of visually rich computer generated imagery for visual effects in movie production. Following in the footsteps of REYES over the past thirty years, this means supporting extremely complex geometry, texturing and shading. In the current generation of renderers, it is essential to support very accurate global illumination as a means to naturally tie together different assets in a picture.

This is commonly achieved with Monte Carlo path tracing, using a paradigm often called *shade on hit*, in which the renderer alternates tracing rays with running shaders on the various ray hits. The shaders take the role of generating the inputs of the local material structure which is then used by path sampling logic to evaluate contributions and to inform what further rays to cast through the scene. We propose a *shade before hit* paradigm instead and minimise I/O strain on the system, leveraging locality of reference by running pattern generation shaders before we execute light transport simulation by path sampling.

We describe a full architecture built around this approach, featuring spectral light transport and a flexible implementation of multiple importance sampling, resulting in a system able to support a comparable amount of extensibility to what made the REYES rendering architecture successful over many decades.

CCS Concepts: • **Computing methodologies** → **Ray tracing**;

Additional Key Words and Phrases: production rendering, spectral rendering, batch shading, movie production, global illumination

ACM Reference format:
Luca Fascione, Johannes Hanika, Mark Leone, Marc Droske, Jorge Schwarzhaupt, Tomáš Davidovič, Andrea Weidlich, and Johannes Meng. 2018. Manuka: A batch-shading architecture for spectral path tracing in movie production. *ACM Trans. Graph.* 37, 3, Article 31 (August 2018), 18 pages.
https://doi.org/10.1145/3182161

1 INTRODUCTION

Computer generated imagery for movie production has always had very particular requirements concerning image quality and feature set of the employed rendering system. Since cinematic movie footage is often shot at characteristic frame rates (traditionally 24 FPS) with wide-open apertures, it is crucial for such a rendering system to support depth of field and motion blur, for instance. The most successful approach to this has been the REYES pipeline [Cook et al. 1987]. In its core, REYES is based on stochastic rasterisation of micropolygons, facilitating depth of field, motion blur, high geometric complexity, and programmable shading.

Over the years, however, the expectations have risen substantially when it comes to image quality. Computing pictures which are indistinguishable from real footage requires accurate simulation of light transport, which is most often performed using some variant of Monte Carlo path tracing. Unfortunately this paradigm requires

random memory accesses to the whole scene and does not lend itself well to a rasterisation approach at all.

That not withstanding, the main challenges in producing the highest quality VFX work today have stayed very much the same as those stated in the original REYES paper [Cook et al. 1987]. Because of that continuity and because of the tremendous success that the original REYES rendering system has had throughout the past thirty years, we want to make some comments about how our experience of these requirements and assumptions [Cook et al. 1987, Sec. 1] has and hasn't changed through the years. Following the order in the original paper:

- **Model complexity.** The observations about visually rich imagery hold up essentially unchanged.
- **Model diversity.** Like the original REYES architecture, we support points, lines, triangles, quads, subdivision surfaces, volumetrics and implicit surfaces. The importance of input from simulation and special geometry such as procedurals, hair, particles and volumes has risen somewhat since the days of the original REYES.
- **Shading complexity.** Programmable shading and lavish use of texturing are still an integral part of movie productions. Today this carries over to using complex, layered BSDFs, potentially featuring spectral dependencies.
- **Path tracing.** Path tracing is now ubiquitous in production rendering, this has also been discussed in [Christensen and Jarosz 2016; Fascione et al. 2017a,b; Keller et al. 2015]. Due to unified handling of all lighting effects and the resulting simplification in workflows, nearly all of the industry has moved over to physically-based global illumination using some variant of path tracing. Given this, it is clear that the assumption that ray tracing has to be kept to a minimum has had the most significant change since 1987.
- **Speed.** While computers have become much faster and symmetric multiprocessing systems with processor counts in the many tens are now ubiquitous, this item still applies largely unchanged. At the same time, scene complexity and detail levels have grown possibly even more than compute capability has, in the face of image resolutions and frame rates having stayed largely the same[1].
- **Image quality.** While focusing more on a new artifact, namely Monte Carlo noise, this requirement is essentially the same today.
- **Flexibility.** In the original REYES paper [Cook et al. 1987] flexibility was rightfully framed as an all-around requirement: many aspects of graphics, from scene representation to image generation were still being invented, discovered and implemented, and it was crucial to allow users to contribute to experimentation and innovation in all these aspects. It is our experience that today it is most important to be able to efficiently implement new path sampling strategies, as these can often have a far larger impact on performance than low-level optimisations.

The analysis of the requirements and many of the main concepts of the original REYES architecture still hold up very well. It is, for instance, largely undisputed that pixel-based micropolygons contribute crucially to the visually rich diversity that is essential for VFX work.

The largest difference between then and now is the essentially ubiquitous application of some variant of path tracing. The advantages are much improved image quality due to global illumination, transparently binding together assets in the scene, as well as simplified workflows due to a reduced number of render passes and intermediate caches (see Fig. 5). While path tracing is comparatively simple to implement correctly, it is notorious for its fundamental lack of intrinsic locality of reference, making it challenging to realise into a system capable of achieving good performance in the dimensions of execution time or space usage. This is in sharp contrast to what was possible with the REYES pipeline, where high performance was achieved via the very means in use today for any high performance computing application: careful leveraging of intrinsic locality, vectorisation and SIMD execution, parallelism, and pipelining.

A number of methods to apply this to path tracing have been devised (see for instance [Eisenacher et al. 2013; Fascione et al. 2017a; Keller et al. 2017]). Our architecture follows a different pattern which allows for more flexibility when it comes to the choice of path sampling algorithm, while still exploiting vectorisation and texture locality during shading. As mentioned above, increased flexibility has proven to be useful for us to achieve fast response times when reacting to new production requirements.

1.1 Main aspects of the Manuka architecture

While there are many detailed aspects about Manuka[2] which deviate substantially from typical approaches, we want to focus on three main structural aspects here. These are to be understood as architectural design principles and the implementation of the renderer mostly follows them, but allows back doors and special cases to support specific production workflows.

As a result of the experience we gathered with our pipeline based on RenderMan and PantaRay (cf. Sec. 2.2), we prioritised the following design goals:

- Cater to extremely complex shading (the übershader used on *Avatar* had approximately 22,000 lines of code).
- Compute pictures of the highest quality, matching live action footage as closely as possible.
- Quickly react to evolving production needs and advances in research.

[1] At the time of writing in 2017, the majority of movie delivery packages consists of frames that are roughly two thousand pixels across (colloquially called *2K*) and are meant to be displayed at 24 frames per second. While higher resolution and frame rates for delivery certainly do exist, they are still considered more as specialty materials. A few years ago it seemed that a higher frame rate might be coming, but the reception from movie going audiences was cold at best. Higher image resolution, roughly four thousand pixels across (called *4K*) seems to be a closer possibility at this point in time, with home video having taken a lead in adoption.

[2] We called our system *Manuka*, as a respectful nod to REYES: we had heard a story from a former *ILM* employee about how REYES got its name from how fond the early *Pixar* people were of their lunches at Point Reyes, and decided to name our system after our surrounding natural environment, too. Manuka is a kind of tea tree very common in New Zealand which has very many very small leaves, in analogy to micropolygons in a tree structure for ray tracing. It also happens to be the case that *Weta Digital*'s main site is on Manuka Street.

These points are implemented in the following architectural corner stones: shade before hit, spectral rendering, and deferred splatting.

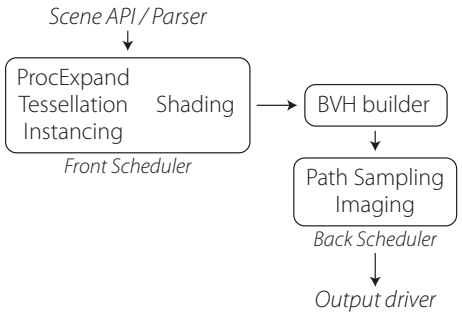

Fig. 2. Coarse overview of our shade before hit architecture. The *Front Scheduler* orchestrates collecting shapes from the input, including procedural expansion, instancing management and tessellating and shading the micropolygon grids. At the end of shading, a BVH is built on these, and the *Back Scheduler* orchestrates light transport sampling the various paths, imaging and communication to the output drivers.

Shade before hit. We use the *RenderMan Shading Language* (RSL) for programmable shading [Pixar Animation Studios 2015], but we do not invoke RSL shaders when intersecting a ray with a surface (often called *shade on hit*). Instead, we pre-tessellate and pre-shade all the input geometry in the frontend of the renderer (see Fig. 2). This way, we can efficiently order shading computations to support near-optimal texture locality, vectorisation, and parallelism. The output of these `prelighting` shaders are the inputs to the light transport stage: per-vertex BSDF inputs. This system avoids repeated evaluation of shaders at the same surface point, and presents a minimal amount of memory to be accessed during light transport time. An added benefit is that the acceleration structure for ray tracing (a bounding volume hierarchy, BVH) is built once on the final tessellated geometry, which allows us to ray trace more efficiently than multi-level BVHs and avoids costly caching of on-demand tessellated micropolygons and the associated scheduling issues.

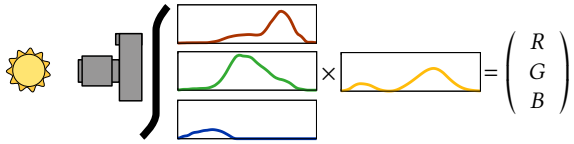

Fig. 3. Spectral rendering forms an image by multiplying the incoming spectral power distribution (shown in yellow) by the spectral camera responsivity curves. As opposed to RGB rendering or using the XYZ colour matching functions, this ensures the rendered images exhibit the same degree of observer metamerism as the camera footage they are intended to match.

Spectral rendering. The light transport stage is fully spectral, using a continuously sampled wavelength which is traced with each path and used to apply the spectral camera sensitivity of the sensor (see Fig. 3). This enables us to faithfully support complex materials which require wavelength dependent phenomena such as diffraction, dispersion, interference, iridescence, or chromatic extinction and Rayleigh scattering in participating media. Also, importance sampling for a specific wavelength can be much more efficient than dealing with a full spectrum or an RGB contribution at once. When it comes to colour reproduction, multiplying path throughput weights in RGB in general produces inaccurate results, as explained for example in [Fascione et al. 2017a; Peercy 1993; Ward and Eydelberg-Vileshin 2002].

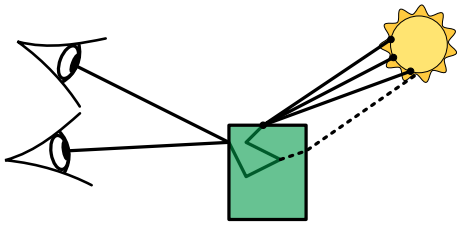

Fig. 4. An example for a cached path with multiple next event estimation samples, one manifold next event estimation sample (dashed), and connections to multiple sensor points (which is useful for instance for stereo or motion blur).

Deferred splatting. Our execution order for path tracing is path-first, that is, a full path is constructed before moving on to the next. Also, a graph structure with complete information about this path, including all path vertices and material interactions at these vertices, is kept in memory before splatting the sample to the frame buffer. Note that this structure is not a linear list of vertices but an acyclic graph. In particular, it potentially includes multiple vertices on sensors and lights if, for instance, splitting, next event estimation [Coveyou et al. 1967], or bidirectional path tracing [Lafortune and Willems 1993; Veach and Guibas 1994] are used (see Fig. 4). This enables us to select different buffers for splatting and to evaluate arbitrary output values for extra channels in a single place after constructing the full sample. We also evaluate weights for multiple importance sampling (MIS) [Veach and Guibas 1995] after the sampling stage, making it easy to dynamically combine arbitrary sampling techniques. This approach also easily supports the implementation of *energy redistribution path tracing* or *gradient domain path tracing* as a post-process.

2 HISTORICAL CONTEXT

In this section, we want to give a quick overview about the train of thoughts and the experiments that lead to Manuka as it is today, surveying previous work on the way.

2.1 Classic RenderMan

In the past, Weta Digital was using Pixar's *PhotoRealistic RenderMan* as the main tool to render images. Increasing demands for realism in particular for visual effects work drove the need to include for instance improved subsurface scattering. This was facilitated using various kind of sidecar caches, which had to be generated in separate dedicated render passes.

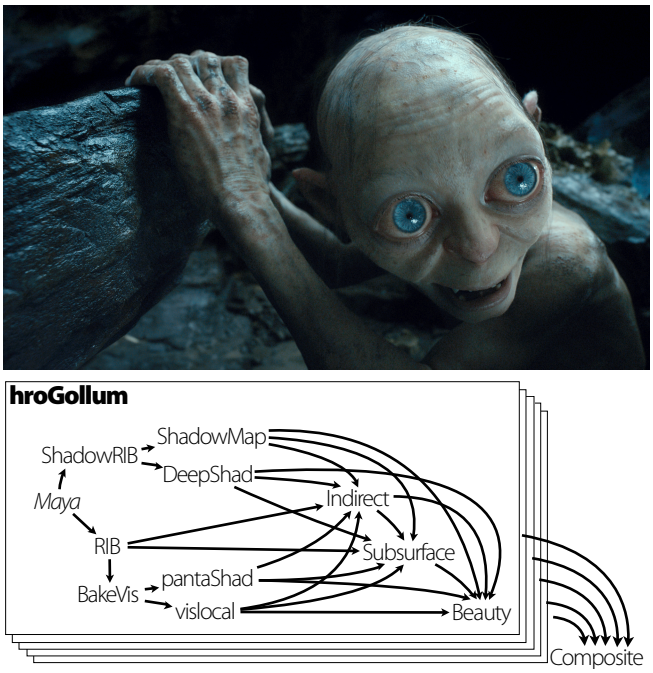

Fig. 5. A frame from *The Hobbit: An unexpected Journey* and an illustration of the corresponding pipeline using RenderMan as backbone. Shadows, local occlusion, indirect diffuse, and subsurface all required separate render passes with intermediate caches as result which would in turn be fed to the final beauty pass. For various reasons (including memory usage), many of such passes would be required and composited to yield a final result. The switch to path tracing promised substantial simplification of this process. . Image courtesy of Weta Digital, ©2012 Warner Bros. Entertainment Inc. All rights reserved

2.2 RenderMan and global illumination

Over the years we added many approximations to global illumination to our RenderMan-based pipeline, such as a point cache-based approach to subsurface scattering used on *Avatar*. To be able to compute more precise occlusion, we employed PantaRay [Pantaleoni et al. 2010] and GPU ray tracing, used to bake weakly directional ambient occlusion for the spherical harmonics pipeline used for instance on *Avatar* [McKenzie et al. 2010] or *The Adventures of Tintin*. Beauty renders were performed using RenderMan at the time, reading the PantaRay occlusion during shading (see Fig. 5 for an illustration). After extending PantaRay to area light sources, god rays [Hanika et al. 2012], and performing first tests with path tracing, we decided to implement a full path tracer to directly produce final images.

2.3 Ray wavefronts

There exists a remarkable variety of approaches to efficiently ray trace complex geometry, diced to the sub-pixel level which is required to obtain sufficient realism.

These include on-demand dicing and sometimes caching of the resulting micropolygons [Christensen et al. 2006; Hou et al. 2010; Lamparter et al. 1990; Pharr and Hanrahan 1996; Pharr et al. 1997; Smits et al. 2000; Stoll et al. 2006] often combined with tracing streams or wavefronts of rays, and ray-sorting methods, through to cache-oblivious methods using sorting of ray intersections instead of caching of micropolygons [Hanika et al. 2010; Moon et al. 2010].

Ray wavefronts are an approach to path tracing in which the ray tracing order is in some sense the transpose of what would result from the textbook description of path tracing: instead of tracing one ray then following it through the path the renderer generates, all first rays for many paths are traced first, then all materials on the hits are evaluated, then all new generated rays are traced and so on, in *waves* of alternating tracing phases and evaluating phases. Shooting wavefronts of rays requires the ability to store many rays along with the state of the associated global illumination algorithm in memory. For large batches of rays, this can quickly become a significant resource commitment.

After experimenting with on demand dicing for a while, we decided on a simpler solution. Our reasoning was as follows: on demand dicing means running shaders to obtain displacements. It was, however, one of our initial design goals to not run shaders more than necessary. This is partly due to the compute required to run shaders, but mostly due to the accesses to many layers of large textures. We wanted to save the memory bandwidth for actual geometry and ray tracing acceleration structure data, as well as being concerned by the load on the I/O infrastructure caused by large amounts of incoherent texture requests.

In our experience sorting rays to yield cache coherence is mostly useful for primary rays. Much of the performance improvements reported for tracing ray wavefronts in conjunction with on demand tessellation and shading (such as in [Hanika et al. 2010]) are likely due to much coarser dicing for secondary bounces. Such aggressive level of detail, however, was not an option for us, given our aim to deliver highest quality images. Indeed it would be easy to construct configurations using non-diffuse materials or shadow silhouettes which would at a minimum require user intervention. Further, this approach is often justified by an argument in which the coarser tessellation is a form of approximate bandlimiting for the signal contained in the incoming radiance at a given point in the scene[3]. However, our observation looking at actual scene contents that are common in VFX production processes (at least at Weta Digital) indicates that the actual problem seems to be that the base cages in our models are actually too dense for this advantage to materialise at a geometric level. Indeed if anything, a lively discussion we've had for years in the industry is centered around methods of automatic mesh simplification that are reliable and efficient enough to complement tessellation in a renderer's geometry processing pipeline. We are aware of this being a sought-after feature on the part of renderer authors and users alike, but it seems this theme has not received much attention from the research community to date. Lastly, depending on the specifics of the texture coordinate mapping technology in use, be that single (u, v) sets, versus (u, v) texture atlases, versus systems like Disney's Ptex [Burley and Lacewell 2008], enlarging the footprint of the corresponding texture lookups may or may not

[3]Note that this is one of those very coarse approximations that trade the integral of a product for the product of the integrals. It is well known how some common use cases might make this tradeoff an unacceptable compromise in quality.

be possible or computationally advantageous, further reducing the potential for bandlimiting to actually occur.

Another unacceptable potential pitfall is the quality of the ray tracing acceleration structure in the case where a top-level tree is built around patches which are to be tessellated and shaded on demand. An extreme example would be hair, where every strand is a separate shape to be tessellated, but has about the same bounding box which goes into the top-level hierarchy, essentially degenerating to a linear list.

Today, this last issue can be alleviated somewhat by partial rebraiding [Benthin et al. 2017]. However, a full BVH build will still yield the best performance. We decided to (view-adaptively) pretessellate all geometry into micropolygons and build one single bounding volume hierarchy on this data. This enables efficient traversal of incoherent rays, and presents consistent geometry to the path tracing layer, independent of ray differential, tracing direction, or number of bounces. This also reduces implementation complexity because there is no need to tweak caches for scaling behaviour.

However, this meant that we would potentially need to go out of core on very large scenes. Because of available literature [Budge et al. 2009; Garanzha and Loop 2010] and our experience with PantaRay [Pantaleoni et al. 2010], we were confident we would be able to handle this.

We first wrote Manuka as a hybrid out-of-core CPU/GPU path tracer using large ray wavefronts (in our implementation, 2 million rays were traced at once). We designed a ray tracing kernel based on swappable pages which used only indices and no pointers. Each page contained a treelet, suitable for coherent memory access during ray tracing, and the required parts of the geometry in the sub tree. BSDF inputs and ray tracing geometry were separated, such that the light transport and geometry intersection phases would be able to make best use of the available memory. The ray tracer supported pausing and resuming rays in mid flight, to enable out-of-core paging.

The complexity of the RSL übershader used for almost everything in the RenderMan pipeline convinced us that we would not be able to run these shaders on ray hit inside the innermost loop of our new renderer. Thus, we decided early on to split the rendering process into two phases, shading and path sampling, an architecture we've been calling *shade before hit*.

As for light transport algorithms, this version of Manuka had support for path tracing with *next event estimation*, *light tracing*, and *Kelemen Metropolis* [Kelemen et al. 2002]. The latter was possible due to a trick that reduced the amount of memory required to store the state of the current samples in the Markov chains significantly: we made use of *multiple try Metropolis* [Liu et al. 2000] to multiplex a few (e.g. 32) tentative samples into one shared current state.

While this simple path tracer was producing pictures, there was clearly room to improve the noise characteristics of the output.

2.4 Manuka today

Soon, more advanced sampling techniques, in particular *manifold exploration* [Jakob and Marschner 2012] and *vertex connection and merging* [Georgiev et al. 2012], made us rethink our architecture. To be able to quickly experiment with these sampling techniques, we wrote a second rendering module inside Manuka, tracing whole paths at once, fully on the CPU. As a surprising result, there was no performance penalty for us as compared to the in-core CPU code path of the wavefront tracer. This may be due to our unique shade before hit architecture: while the rays connecting directly to camera are still coherent, there is usually very little common memory traffic to be extracted from indirect diffuse rays, increasingly more so for longer paths. The way we extract BSDF inputs from per-vertex data also does not require any texture filtering, which might have allowed for some caching effects on larger texture tiles. Further, the ray-sorting step and some memory traffic accessing the ever growing ray states disappeared.

On the other hand, the increased flexibility in choosing and crafting interesting sampling strategies yielded a handsome payoff. This may be no surprise, as it is well known how the rendering error in Monte Carlo sampling is proportional to the ratio of the sample standard deviation σ to the square root of the sample count N. That is to say that in a time-to-equal-error setting, a new technique which halves the standard deviation can take up to four times longer per sample before breaking even with drawing more samples from the old technique. The conclusion is then inevitably that optimising for variance reduction is more important than optimising for faster generation of samples.

When it comes to the ray tracing kernel, the choice to trace single rays simplified our requirements a lot: it meant we do not need support for ray interruptions, ray-sorting, or manual paging of ray tracing data. Instead, we are now facilitating out-of-core in case it becomes necessary by simple memory mapping of the data backing. This choice was motivated by our experience with how users were working with PantaRay for area light sources: due to non-local memory access patterns, the cost for going out-of-core in this case was so much higher than for the original application (computing local occlusion), that users generally tried to minimise geometry or pick a larger machine and would only rarely resort to out-of-core.

Moving on to newer versions of Manuka, the GPU code path became less important for us, mostly due to a combination of memory constraints on currently available hardware compared to our production scene sizes, and maintenance issues when deploying to a new GPU architecture. Since temporal consistency is of the utmost importance to us, we replaced the Metropolis code paths by specialised sampling techniques working from within pure Monte Carlo, such as moving from *manifold exploration* to *manifold next event estimation* [Hanika et al. 2015].

One goal of the Manuka project was to build a flexible, modular architecture, which would make it easy to replace modules of the renderer to experiment with new ideas and try different approaches. Indeed, this short summary of Manuka's history shows that we did in fact exchange a number of our modules multiple times. In contrast, the data flow from RIB input to the path tracing backend, including the *shade before hit* architecture was there from the start. Also, light transport has always been spectral and subsurface scattering was fully path traced from the beginning.

3 ARCHITECTURE

Pipeline Overview. Manuka comes with a relatively complete implementation of a classic RenderMan frontend. This includes a RIB parser and support for RenderMan Interface Filter plugins, procedural geometry plugins, shadeop plugins and dspy plugins as

described in the manual for Pixar's PhotoRealistic RenderMan versions based on REYES [Pixar Animation Studios 2015]. A few extensions have also been implemented, notably a procedural geometry interface to implement plugin tessellation engines, which turned out to be very useful for Level of Detail procedurals. Micropolygon grids can then be obtained either from RIB primitives through Manuka's tessellation engine, or directly from these tessellation plugins. Manuka's internal tessellation engine is a modern implementation of REYES *split-and-dice*, supporting polygon, NURBS, subdivision surface, curve, implicit surface and volumetric dicing, with the implementation features one would expect such as parallelisation and vectorisation. Shadeable micropolygon grids obtained as above are then processed through the following pipeline:

- First, grids from multiple faces that are small enough are joined into *multigrids*, to improve the efficiency of the SIMD execution and texturing subsystems. Note that this step takes care of deduplicating vertices on boundaries which will then be shaded only once. The layout and semantics of the binding of *primitive variable* to shader parameters effectively implies that multigrids cannot be formed from grids that disagree on the values of non per-vertex data (corresponding to the `uniform` and `constant` storage classes in RIB), such as references to multiple textures out of a texture atlas. Although this might sound like an impediment, it is actually a desirable property to have from an access coherence point of view.
- Multigrids are dispatched to various execution threads and shaded in parallel. Manuka shaders are written in our implementation of RSL 2.0 [Pixar Animation Studios 2015], compiled down to native machine code and executed in a SIMD fashion as required by language semantics.
- Running shaders on multigrids results in producing per-vertex BSDF input data which is then transferred to the BVH builder in the backend. Out-of-core is performed by using memory mapping on this storage.

After all grids have been processed and collected, a bounding volume hierarchy is built on the complete list of micropolygons (except for instanced geometry, which has separate BVH and transformation matrices). In the second phase, light transport takes place, and a multitude of supported techniques can be used to construct transport paths. Once a complete path is constructed, it can be evaluated and the contribution added to the corresponding pixels as specified by the configuration of the various arbitrary output variables.

3.1 Shade before hit

We decouple shading and path sampling. That is, evaluation of RSL shaders happens in batched mode on all vertices of the tessellated micropolygons before the light transport phase starts. While in classic RSL the result of running a surface shader is the final color at that location, and in OSL [Gritz 2009] is a *closure*, in Manuka the result of a shader is a data aggregate containing the BSDF inputs, their weights, and the layering setup: once shading ends, this aggregate is compressed and stored as per-vertex data along with the geometry. We keep ray tracing geometry and BSDF input data separate to improve locality of reference during ray tracing: the separation helps ensure that shading data will not be accidentally fetched into the processor cache during geometry intersection computation.

Differently put, we evaluate the loop invariant only once, and convert primitive variables to BSDF inputs for each vertex on a grid. Since the compressed BSDF inputs are much smaller than the set of input textures used to shade a vertex, this reduces the memory pressure during path tracing (as well as avoiding the potential for I/O stalls from texturing). Moreover, this approach avoids repeated shading of the same grid vertices when intersecting the same micropolygon multiple times during the tracing of different paths. This especially pays off for high sample counts and long paths (that is, paths with many bounces, such as in hair for instance in Fig. 6).

As opposed to the original REYES paper, we use bilinear interpolation of these BSDF inputs later when evaluating BSDFs per path vertex during light transport[4]. This improves temporal stability of geometry which moves very slowly with respect to the pixel raster, especially for coarse tessellations. Note that this only applies to the BSDF inputs: the BSDF itself is evaluated exactly at the hit location (in other words: the hit is Phong shaded, based on parameters that are Gouraud interpolated).

Since shading happens once on startup for the whole grid of tessellated vertices, our pipeline inherits many of the properties described by Cook et al. [1987, Sec. 2.3] when it comes to texture locality and vectorisation: texture locality is enforced by merging grids up to texture seams before shading. Internally, the system uses a texture cache which easily achieves very high access coherence figures: being able to use the texture coordinates traveling with the grids to schedule shading execution means that only projection-based texturing requires a proper texture cache[5], because the scheduler can easily dispatch execution in a texture coherent manner for (s, t)-mapped grids. To make sure vectorisation makes use of available SSE or AVX capabilities when running renders on an heterogeneous machine room, we *just-in-time* compile shaders from LLVM bytecode on renderer startup (with the aid of a cache of precompiled binaries to save on redundant recompilations).

We follow this paradigm for hair strands and volumes as well, and perform view-adaptive tessellation into line segments and voxels, respectively, followed by shading during ingestion of the input. On hair, this enables the artists to add fine details and displacement for increased realism.

3.2 Spectral rendering

We simulate light transport by modelling colours as spectral power distributions. This choice has large advantages over simply using tristimulus values in terms of accurate colour reproduction as exemplified for instance by Fascione et al. [2017a].

Colour formation. Reproduction of colours from spectral data is a fairly well understood process [CIE 1996; CIE 2004; ITU 2002; Wyszecki and Stiles 2000]. One important aspect when working

[4]We note a very similar capability has been available for the longest time in Pixar's PhotoRealistic RenderMan and is part of the RenderMan Interface specification at least since version 3.2.

[5]We note that separate assets sharing textures may also benefit from an appropriately large texture cache, but we also observe that such a cache would need to be conspicuously large in practice to withstand the load of dozens of threads concurrently shading assets each accessing dozens to hundreds of individual files.

Fig. 6. A frame from the movie *War for the Planet of the Apes*. This shot includes several characters with fur as well as multiple volumetric light sources and foliage. A special challenge in this render are the tiny moss structures on the ground. These are modelled to very high detail, resulting in sub-pixel base meshes. This presents a hard case for a *shade before hit* architecture: since there are many more vertices even on the base mesh than pixels. This is a sub sampling situation with many more shaded vertices than materials constructed for ray intersections for low sample counts. Image courtesy of Weta Digital, ©2017 Twentieth Century Fox Film Corporation. All rights reserved.

with spectral data is to pay due consideration to the phenomenon of *observer metamerism*, in which radiation from different spectral distributions appear to have the same colour to a given observer, see for example [Lam 1985] for an extensive discussion of the subject.

The commonly recommended approach to convert a given spectral distribution to tristimulus values would be to project the distribution from the current sample into XYZ space and then using an appropriate transformation matrix to transform from XYZ coordinates to the desired colour space (common colour spaces used in practice include *sRGB* [Stokes et al. 1996], *AlexaWide* and *ACES* [Agland 2014]). Indeed most contemporary definitions of colour spaces employ XYZ coordinates in this central role of pivot space.

Naturally, and by definition, projection into the XYZ coordinate space amounts simply to computing the scalar product of the given distribution with the colour matching functions $\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda)$ defined by the CIE in 1931, and derived from the Wright/Guild data [Fairman et al. 1998; Smith and Guild 1931; Wright 1928] for the so-called *two degree observer*[6].

A notable trait of this approach is that the transformation obtained using the Wright/Guild data has the same null-space (that is, the space of metamers) as the human visual system (this is because the Wright/Guild LMS space[7] and the XYZ space are isomorphic and differ only by an invertible 3×3 transform). This is of course by virtue of careful design choices, and constitutes a very desirable property if the intent is to store a scene's colour so that it matches the impression a human would have when observing said scene in real life.

In our use case, though, the most important trait is actually not to match the sensation observers would have in real life were they present in the scene, but instead it is for the rendered images to match the metamerism that the motion picture camera exhibited when the plates were captured, so that the integration of rendered and captured data is as seamless and accurate as possible. For this reason we employ spectral responsivity curves during rendering that are built measuring the spectral sensitivity of the principal photography camera and transforming these measurements into colour matching functions to produce tristimulus data in XYZ 1931 colour space coordinates. Such spectral sensitivity curves can be measured [Hardeberg et al. 1998] or derived approximately [Jiang et al. 2013; Kawakami et al. 2013]. There exist commercial devices for such measurements [Image-Engineering 2010] as well as do-it-yourself kits [Karge et al. 2014] for home-made devices. We are actively working on developing an improved device for camera characterisation to be used in the future.

[6]Later experiments on sensitivity of human subjects to colour sensation revealed that the apparent angular size of the source influences the ability of the observer to perceive and distinguish colours, due to the different distribution of cones and rods in the foveal and non-foveal regions of the retina. A second set of measurements, meant to better embody the concept of a *standard observer* was then taken using a larger source spanning about ten degrees (sometimes called the *supplementary standard observer*). This not withstanding, standard colour spaces (XYZ, Rec. 709, Rec. 2020, ..) are still referred to the original two degree observer data from 1931.

[7]The space called LMS is the space of sensitivity of the cone receptors in the retina: these are classified as *L-cones*, *M-cones* and *S-cones* due to their sensitivity to long, medium and short wavelengths.

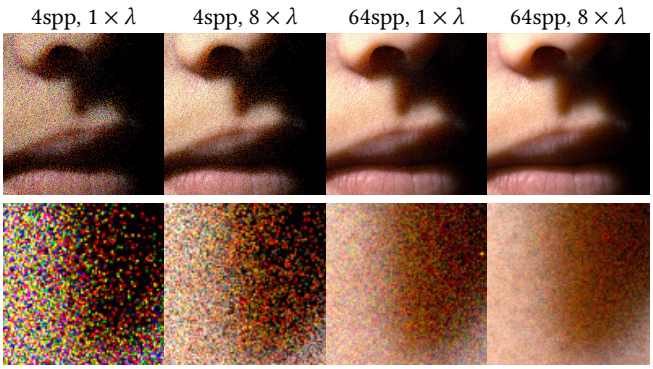

Fig. 7. Closeup view of a skin render with wavelength dependent extinction under the surface, using path traced sub surface scattering. The images labeled $8 \times \lambda$ use exactly the same paths as the $1 \times \lambda$ counterparts, but also evaluate them for additional wavelengths. Individual contributions are combined using multiple importance sampling [Wilkie et al. 2014]. This helps reduce colour noise. Thanks to the *wikihuman* project (http://gl.ict.usc.edu/Research/DigitalEmily2/) for the asset shown in these images.

Spectral transport. In general, rendering using directly the tristimulus values of an RGB colour space yields non-physical results [Ward and Eydelberg-Vileshin 2002] and a different working space will yield different results [Agland 2014]. This observation, combined with the quest for the best possible importance sampling as well as a requirement for precise matching to camera footage from principal photography, motivated our choice to perform spectral light transport.

Spectral rendering can be performed in many ways (see for instance [Maloney 1986; Meyer 1988; Peercy 1993]), but in the interest of best importance sampling we chose to trace a single, randomly selected wavelength for each path. Every additional random variate introduced in a Monte Carlo method will introduce additional noise. To control the resulting colour noise, we employ a stratified sampling scheme called *hero wavelength spectral sampling* [Wilkie et al. 2014], which provides a good opportunity for employing SSE instructions to compute the throughput of additional wavelengths simultaneously at little additional cost. A visual comparison can be seen in Fig. 7.

Upsampling of RGB textures to spectra. While we support using full spectra from binned data as input, we can save memory cost by using RGB textures. It is possible to upsample traditional RGB values (or any colour managed tristimulus input) to a full spectrum suitable for spectral rendering [MacAdam 1935a; Meng et al. 2015; Smits 1999; Sun et al. 1999; Wang et al. 2004].

In general, not all input chromaticities will result in energy conserving reflectance spectra [MacAdam 1935b; Schrödinger 1919]. This is because highly saturated chromaticity relates to a peaky spectrum, but the maximum is limited to one. This in turn means the brightness, which is related to the integral of the spectrum, needs to decrease. As a consequence, some kind of gamut mapping [Finlayson and Hordley 2000; Forsyth 1990; Morovi 2008] needs to be performed to output energy conserving spectra with close chromaticity. A number of strategies to achieve this for reflectance spectra have been described by Meng et al. [2015].

Brighter colours with high saturation [Couzin 2007] would require the renderer to simulate fluorescence during light transport time [Glassner 1994]. We decided not to implement fluorescence at this point because it is unclear how to make bidirectional path construction efficient.

In Manuka, spectral upsampling uses an optimisation strategy similar to the method proposed by Smits [1999]. There are some important differences though. First, our optimisation converges even for a much higher output resolution. For emission spectra, we allow the distribution to grow above one (there is no constraint on energy conservation), and instead constrain the process such that a D65 input spectrum round-trips without error, i.e. it can be converted to CIE xy chromaticity coordinates and back to a full spectrum. For both emission and reflectance spectra, the optimiser uses the specific input colour space of the texture instead of using fixed sRGB primaries. For reflectances, we also use CIE Illuminant E as white point, since it makes sense to expect a uniform reflectance spectrum when the input has equal $R = G = B$ coordinates: after all, the white point is governed by the illumination, which is not part of the reflectance spectrum. Also, we make use of the spectral camera responsivity curves in the process instead of using the XYZ colour matching functions: to determine when a tristimulus colour and a spectrum are equivalent, we determine both in camera RGB using the spectral response curves of the device. For image based lighting, it is most meaningful to employ the responsivities of the camera that was used to take the picture.

Materials and importance sampling. Spectral rendering allows us to easily support rich material models with very high degrees of accuracy, including effects such as far-field diffraction, interference, iridescence (cf. Fig. 8), or Rayleigh scattering in participating media. One important feature for us is that we can robustly handle coloured extinction in participating media. Rendering this otherwise may quickly lead to infinite variance [Raab et al. 2008], even when using rejection sampling by throughput [Szécsi et al. 2003], or leads to inefficiencies when conservatively sampling the shortest free path of all wavelengths [Kutz et al. 2017]. Sampling a path specifically for a wavelength provides better importance sampling, which is essential to render high-quality skin by path traced subsurface scattering.

Photometry. We make use of colour management throughout the pipeline, employing photometric quantities [USAS and ASME 1967] to support more intuitive user interfaces for lighters: this enables them to change the colour of a light without affecting perceived brightness, for instance.

3.3 Path first execution order

We decouple path sampling from MIS evaluation and splatting: the path sampling techniques are only responsible for implementing path construction and inserting every created vertex into an acyclic vertex graph. In particular, since the MIS weights are computed separately, the techniques do not need to be informed about any other potentially active techniques.

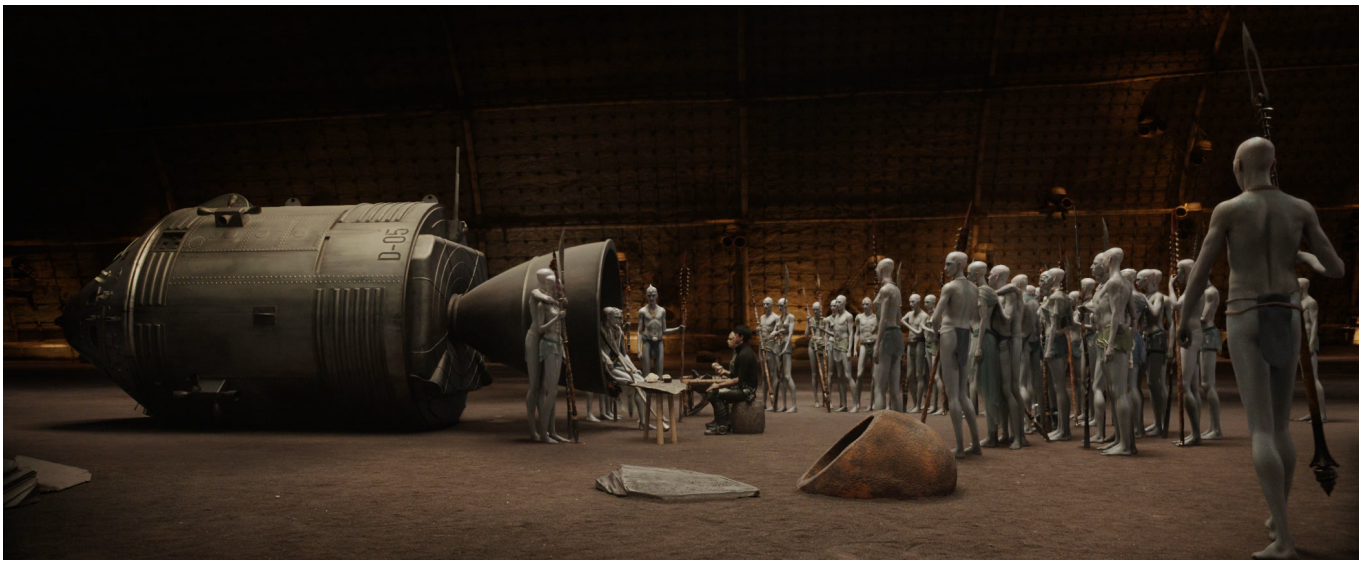

Fig. 8. A frame from the movie *Valerian and the City of a Thousand Planets*. The characters feature a complex multi-layer iridescent skin shader, making them heavy-weight mostly to evaluate the BSDF at hit time. On the other hand, the iridescent appearance is simple to implement in a spectral renderer. This frame includes a few plate elements: the backdrop, the space ship, and the two human main actors. Image courtesy of Weta Digital, ©STX Films and EuropaCorp. All rights reserved.

The completed group of paths is then handed over to the next phase, where the probability density of all active techniques is evaluated, to compute final MIS weights for the constructed paths in the graph. Note that the vertex graph caches all relevant quantities along the paths including potentially expensive BSDF and PDF evaluations. This results in efficient computation of the MIS weights, while avoiding the complexity of recursive weight evaluation during path construction [Georgiev 2012]. This supports one of our main design points, increased flexibility, as it avoids the complexity of the implementation growing too quickly when new sampling techniques are added. In practice this means for instance that we can easily combine path tracing with specialised estimators for single scattering, such as in Fig. 9.

A disadvantage of our approach is that the irregular structure of the vertex graph cache occasionally requires the per-thread data structure to grow dynamically and hence requires extra care on memory usage to remain efficient. Also, we need to keep complete material instances around for each vertex. While the overall memory consumption for a few hundred threads is negligible when compared to geometry and BSDF inputs, this approach would be too memory heavy for a ray wavefront implementation where ray counts are in the millions.

Using this decoupled approach to path sampling, we can easily support all kinds of arbitrary output variables, which can be computed on the full path after sampling is finished. More advanced secondary sampling techniques such as *gradient domain path tracing* [Kettunen et al. 2015] or *energy redistribution path tracing* [Cline et al. 2005] or our custom path reconnection technique optimised for stereo, motion blur and depth of field can be run as post process on the path vertex graph cache, without complicating the rest of the path space code.

During splatting we use an approach inspired by *density-based outlier rejection* [DeCoro et al. 2010], also filtering out firefly samples in the process.

3.4 Selected subsystems

In the following, we review a few of our subsystems that while not directly constituting the main architecture of Manuka, follow individual design decisions that tie into the system around it.

Dicing. The input geometry is passed through the tessellation pipeline to create micropolygon grids which are then shaded and stored until the time comes to build a BVH for ray tracing. The main goals of the tessellation pipeline are accurate dicing to achieve good matching to the specified shading rate, watertight tessellation and a good grid parameterisation for derivatives and differentials.

The tessellation engine supports a number of primitive classes of various conceptual geometric dimension: points, curves, polygonal meshes, subdivision surfaces, various kinds of bicubic patches such as NURBS and Beziér patches, implicit field surfaces and volumetric primitives. Naturally, some of these high level primitives allow the corresponding tessellators to be particularly simple: a micropolygon grid generated from a points primitive is simply a set of nearby points, all of which are shaded and then go into storage representing either oriented disks, ray-facing disks or geometric spheres. Similarly, the curves tessellation logic dices the curves down the length to produce grids that are either an appropriately sized array of short one-dimensional segments (which again can be oriented, ray-facing or true cylinders with various kinds of "elbows") or strips of vertex-pairs to represent oriented segments as surface micropolygons.

Fig. 9. A frame from the movie *Guardians of the Galaxy Vol. 2*. This shot is particularly heavy on procedurally created fractal geometry and participating media. Decoupling path sampling and MIS evaluation allows us to easily mix and match many different specialised importance sampling schemes. It presents a hard case for our architecture because of the vast amount of finely detailed geometry, which could potentially benefit from occlusion culling. Image courtesy of Weta Digital, ©2017 Marvel. All rights reserved.

For what concerns polygonal meshes and subdivision surfaces, which are the vast majority of the primitives encountered in our scenes, we proceed in a manner reminescent of contemporary GPU tessellation units: we do not tessellate faces to a regular lattice, but instead can create an arbitrary number of vertices on each edge, as this enables the tessellator to produce watertight micropolygon meshes without the need for stitching[8]. Our algorithm is inspired by *DiagSplit* [Fisher et al. 2009]: tessellation happens adaptively using an iterative multi-level approach, starting with the base cage as first level. At each level all active patches cast a vote on tessellating their edges. Then the votes are arbitrated and a final per-edge decision is reached. After that, the patches decide whether they can dice or need further splitting. Those that need splitting go to the next level.

This stage was no performance bottleneck for us, since we run it once up front and it is relatively light-weight compared with shading. If needed, however, this scheme can be (but currently is not) parallelised: all patches cast their votes in parallel, then all edges are arbitrated in parallel, then all patches make their decision in parallel.

[8]Some tessellation architectures rely instead on the so-called *stitching* process: when two adjacent micropolygon grids are tessellated so that the vertex counts on the common edges don't match, a strip of *stitching triangles* is inserted in place. Note that this can be an extremely efficient way to fix pin-hole issues in rasterisation architectures, because these primitives are small, relatively easy to sample and require no extra shading computations to happen. However, we preferred to avoid the injection of a large amount of small triangles with poor aspect ratios into our BVH, to reduce the risk of poor numerical behaviour during intersection testing.

Displacement may cause cracks at patch boundaries, due to differences in texture filtering during lookups in the displacement texture. In order to deliver watertight meshes to the geometry storage, we use a welding pass after displacement: we can uniquely and consistently enumerate all vertices on patch edges, since the tessellation scheme makes sure that every edge has the same number of vertices, regardless of the face. The position of matching vertices are then analyzed and reconciled, and the two vertices are then moved to the new positions, while normals are kept unchanged to retain sharp edges and creases. Note the vertices are not deduplicated.

As we use RSL for shading, we also support arbitrary differentiation. In some renderers inspired by REYES, the tessellator produces reliably rectangular grids, making the implementation of derivatives a relatively straightforward application of central differencing. In our case, the non-rectangular topology of our dicing scheme results in grids comprising a mix of quadrilateral and triangular micropolygons, requiring a somewhat more careful selection of neighbouring vertices and weighting to achieve appropriately high numerical stability in the derivatives. The interesting counter to that turns out to be a nice pay-off in terms of reduced aliasing when differentiating signals that run near-aligned to a grid's (u, v) intrinsic parametrisation.

Dicing oracles. The dicing oracles instructs the tessellator to dice the primitive into micropolygons (or microvoxels for volumes) of a certain size. We adopted in Manuka the RenderMan notion of

shading rate [Cook et al. 1987; Pixar Animation Studios 2015]: the shading rate represents approximately the number of pixels a micropolygon covers on screen[9]. By default, we use a shading rate of 1 for front-facing micropolygons, wheras backfacing ones use 10, and out of frustum use 100. We have found it useful to implement a number of specialised dicing metrics for perspective cameras, auxiliary passes for purposes such as texture flattening, and uncommon projections such as fisheye or latlong cameras. Among other inputs, the dicing oracles use one or more camera objects to orient the tessellation metric as needed. This usually works well for multiple output cameras, such as for stereo renders.

Ray tracing acceleration structure. To accelerate ray tracing, we use a nearly standard binary bounding volume hierarchy, built using the surface area heuristic [Aila et al. 2013; Goldsmith and Salmon 1987; MacDonald and Booth 1990]. For motion blur, we support variable $2^k + 1$ time samples per vertex, which turn into the same number of linearly interpolated bounding boxes per node in the tree. This is loosely inspired by arguments outlined by Grünschloß et al. [2011]. These motion boxes are combined when two nodes are merged into a parent node. We support a maximum of $k = 6$ which turns into 64 motion segments per shutter interval. If more precision is needed (for instance for a quickly moving rotor), we use transformation blur, which is conceptually similar to instancing with time-dependent transformation.

Traversal uses SSE instructions for bounding box interpolation for the ray time, and employs an approach inspired by Intel's Embree ray tracing library, where the 3×4 ray/plane tests used to intersect one ray with two bounding boxes are turned into three SSE tests.

Our geometry is stored along with the bounding volume hierarchy in a compact encoding, organised as pages of treelets, with vertices sorted by Morton code for increased locality.

The ray caster is wrapped in functionality to support nested volumes by discarding intersections with lower priority [Schmidt and Budge 2002], to handle transparency for occlusion rays, and to discard intersections which need to be excluded for production functionality such as holdouts.

Ray tracing precision. To avoid surface acne [Woo et al. 1996], many rendering systems apply some sort of small ray bias ε when starting a new ray from a shading point on geometry. This ε is usually chosen proportionally to the floating point value of the world space position of the hit point (with the aim of only changing the mantissa by a constant amount, not touching the exponent if possible). We apply a variant of mail boxing instead, and equip each new ray with a small, ε-sized bounding box inside which the same primitive id shall be ignored[10].

Another severe problem are missed intersections. Especially rays tunneling through the edge between two micropolygons can result in clearly visible light leaking or other similar artifacts. To address this we use a variant of the triangle intersection test described by Chirkov [2005]. As outlined by Hanika [2011, Sec. 5.2.6], this test can be implemented to err on the safe side: the idea is to code the intersection test so that the triangle is effectively enlarged by the uncertainty caused by floating point quantisation, a similar concept is applied very successfully to bounding box intersection testing by Ize [2013]. After some experimentation with this approach, we decided to replace it with a different modification, which does not make the micropolygons bigger, but instead guarantees we get a consistent answer to the ray/edge sidedness test for micropolygons, independent of which side of the edge is being tested against a ray. This approach can be usefully paired with our intersection test for bilinear patch micropolygons, which is a similarly adjusted version of the test described by Ramsey [2004].

To increase floating point resolution where it's most needed, it has proven useful to represent geometry in a coordinate space with its origin close to the render camera positions. This is related to the concept of the `worldcamera` object in RenderMan, in that users can specify in RIB what space to use for micropolygon representation by simply defining a camera object of such name.

Materials. Our material system is designed for maximum realism by implementing complex physical processes as well as supporting fine details.

This includes fully path traced subsurface scattering using the Dwivedi sampling scheme [Křivánek and d'Eon 2014] for dense media such as skin. We support a set of diffusion profiles along with ray traced sub surface [Christensen and Burley 2015; D'Eon and Irving 2011; Frisvad et al. 2014; King et al. 2013], too, but in general the quality/performance trade off is in favour of the fully path traced model.

Our BSDFs are composed of a layering system, modeling interreflections to a user-specifiable degree of realism. This run-time evaluated stack is more expensive to evaluate than just querying a precomputed texture (such as Burley's *Disney principled BRDF* presented by McAuley et al. [2012]), but fits our overall architecture well: in general, our renders tend to be memory bound, so trading compute for memory accesses is sometimes profitable for us. The main reason behind using a layering system that can depend on the incoming ray direction is however much increased visual quality.

Since we have a spectral renderer, we can easily include advanced spectral effects [Weidlich and Wilkie 2009] as can be seen in images from the movie *Valerian and the City of a Thousand Planets*.

We support near- and far-field models for fibers and for sub surface scattering, the latter switching to diffuse albedo and translucency when viewed from far away. Our fiber model supports eccentricity (since *Furious 7*) as well as seamless transition from far-field pre-integrated curve scattering to a near-field model (first used on *The Jungle Book*). We also employ a level of detail mechanism that turns many thin hairs into one thicker but more transparent hair (first employed on *The Hobbit: The Battle of the Five Armies*). This helps control variance for sub-pixel wide hairs in the distance.

To facilitate ray differentials which depend on both geometric curvature and BSDF blur, the material system provides a unified roughness estimate for all interactions to the light transport algorithms, based on the mean cosine.

[9] The naming is RenderMan convention, and it has been argued at first that *tessellation rate* would have been a better name. Given the fundamental coupling of micropolygon grids and shading in REYES though, the tessellation rate does have a $1:1$ correspondence to the rate at which shading actually is executed, as all micropolygon vertices must be shaded, thereby justifying the name.

[10] As discussed in the section on dicing, not all of our micropolygon primitives are flat: for bilinear patch micropolygons, for example, there could be two valid intersections between a ray and a single micropolygon.

Volumes. We use a data structure that is very similar in spirit to Pixar's [Wrenninge 2016] to support efficient motion blur. We treat volumetrics as a first class citizen, and encourage using it together with surface transport. We follow the *shade before hit* tessellation paradigm for volumes too: we adaptively and anisotropically dice the volume depending on projected pixel size and depth. Note that this is not a perspective grid, but a spatial subdivision scheme driven by the dicing oracle, deriving depth from the coordinate system of a designated camera. The depth dimension is simply not subdivided as often, such that the scheme carries over transparently to out-of-frustum areas and multi-view renders. This not only results in less overall data, but also lends itself well for regular tracking [Sutton et al. 1999] (i.e. a 3D DDA), where all voxels pierced by a ray are traversed deterministically. In particular, this makes sure that every voxel is touched exactly once, whereas null collision-based trackers (such as Woodcock tracking [Woodcock et al. 1965] or residual ratio tracking [Novák et al. 2014] to sample distances or to estimate transmittance) are often less efficient, since these trackers are likely to ask for collision coefficients inside the same voxel multiple times. Also, regular tracking enables unbiased support for more complex rendering algorithms with strict requirements on PDF evaluation, in particular the hero wavelength sampling scheme. This is important for us in media with chromatic extinction, such as skin.

We note that with *spectral tracking* [Kutz et al. 2017] there exists a method to exploit spectral correlation of samples even for null collision-based tracking. It is, however, fundamentally incompatible with the hero wavelength sampling scheme because spectral tracking cannot provide PDF evaluations.

Our volumes can be overlapping in many ways: In the most common case, multiple (potentially instanced and heterogeneous) collision coefficients are summed up to form the final medium. This is equivalent to summing the densities (for equal scattering cross-sections) or multiplying the transmittances. Sampling a distance proceeds by sampling one tentative distance per volume and selecting the shortest one. Computing the PDF is particularly simple when all volumes implement distance sampling by transmittance: the result is the product of transmittances and the normalisation constant is the sum of the extinction coefficients. However, we support a few more specialised sampling strategies such as building arbitrary CDFs, forcing scattering vertices before holdout objects, or biasing random walks for Dwivedi sampling. The general case requires to compute the probability that the other volumes did not sample an event before the given distance, which can be more costly. Volumes can further be enclosed in watertight shapes, and will potentially be collected along a ray segment through multiple transparent interfaces. To resolve multiple overlapping shapes with internal volumes, we do on-line constructive solid geometry using user-defined volume priorities [Schmidt and Budge 2002].

Finally, we also support collection of emission from very thin media by including line integration [Simon et al. 2017].

Path space sampling. We implemented and experimented with a wide variety of path sampling algorithms. Still, the most frequently used technique to render an image remains path tracing (from the eye) with next event estimation. Part of the reason may be that we implemented a fairly full-featured *light hierarchy* to sample high-quality light source positions for next event estimation [Keller et al. 2017; Pharr et al. 2017; Shirley and Wang 1991; Walter et al. 2005; Wang and Åkerlund 2009]. We further improve path tracing results by adaptive sampling in image space, using variance estimation. This technique works best in conjunction with unidirectional techniques starting at the eye. As path space adaptive sampling, we employ guiding [Vorba et al. 2014]. To efficiently cover a set of important caustic paths, we use manifold next event estimation [Hanika et al. 2015], which is a stripped down Monte Carlo variant of manifold exploration [Jakob and Marschner 2012]. We also implemented vertex connection and merging [Georgiev et al. 2012; Hachisuka et al. 2012]. This technique, however, is very costly due to the many techniques that have to be considered in the multiple importance sampling mix, and is used only in absence of alternatives. Another problem with this technique is that the photon kernel estimation comes with bias that may lead to visible quality degradation. The same is true for beam radiance estimates, which we have as a tool, but do not use frequently. All variants of Markov chain methods have not found adoption in production so far, mainly because of uneven convergence properties due to autocorrelation of the samples in the chain. In volumes, we use an approach very close to equi-angular sampling as presented by Kulla and Fajardo [2011], and build a custom CDF used for sampling a combination of phase function, transmittance and the distribution of incoming light, as well as forcing scatter events before holdout objects (by renormalising the probability distribution to a fixed maximum).

While not all techniques from academia were applicable to production work, generally more expensive but higher-quality sampling has paid off for us. In the interest of generating predictable results, we prefer to improve the sampling over approximations. This means for instance using Dwivedi sampling [Křivánek and d'Eon 2014] over diffusion profiles, better importance sampling in the light hierarchy instead of clustering or light cuts approaches [Walter et al. 2005], and to avoid using photons and beams.

Geometric ray differentials are used for path guiding lookups as well as for photon map lookups in VCM and are implemented as a side product of the constraint derivative computation for manifold next event estimation.

Light source sampling. Manuka had to scale to millions of light sources since the early days, because of the way we model emitters: in our system light sources are simply geometry that happens to have one or more emissive lobes. This means we can intersect them by path tracing, but tessellating and shading them as regular geometry also transparently results in support for textured and displaced lights. On the other hand, this also means we have to efficiently handle millions of emitting micropolygons. To facilitate this, we build a hierarchical structure on top of the tessellated lights, similar to [Walter et al. 2005; Wang and Åkerlund 2009]. The hierarchy supports a variety of query types. We mainly select an emitting micropolygon by using an estimate of radiance incident to a given shading point.

Similar to Donikian et al. [2006], we use a screen space cache to refine the sampling probabilities. In particular, we learn which nodes in the tree yield successful visibility tests for certain regions

in screen space. This is done by tracking an average correction term, which relates sampling density to actual radiance in the evaluation. Note that this also corrects other potential shortcomings in the sampling density, not only visibility.

We also support selection via estimation of BSDF values, which we limit to primary path vertices only given its moderate cost. The screen space cache is also leveraged to accelerate this computation; the variance in the BSDF evaluation of nodes is tracked and, once it falls below a threshold, a cached evaluation is used instead. This happens when either the roughness of the BSDF is high enough that its evaluation is similar in all directions, when there are no nodes in the hierarchy that lie within a cone of directions where the BSDF varies much, or when the sub tree being evaluated lies within a cone of directions small enough that the BSDF evaluation remains close to constant.

In the case of volume scattering, the light hierarchy supports a special query type, which takes a long beam as input, and returns a light source sampled proportional to a measure of how much the light contributes to that beam. This is used to accelerate single scattering or to build more accurate CDF for free path sampling. If the volume within which sampling is occurring is homogeneous, then we further evaluate the phase function in order to importance sample relevant light sources.

Image space sampling. While the techniques mentioned in the previous paragraphs greatly reduce the variance of samples that land on the image, it remains true that for most production scenes, local regions of the final image will have widely varying amounts of noise present in the incoming signal. For example, a character with complex shading and subsurface scattering effects will generally produce noisier samples than a nearby flat wall that is directly lit without complex occlusions. Using uniform sampling of the image space for such scenes would either result in a fast render where parts of the image remain unusably noisy, or a slow render where effort has been expended needlessly sampling regions of the image that are already converged to a qualitatively sufficient degree.

Manuka's implementation of adaptive sampling (see [Zwicker et al. 2015]) tracks a per-pixel estimate of the variance of the sample luminance, which is updated on-line during splatting after the reconstruction filter is applied. We perform classical reconstruction filtering insofar as we do not perform filter importance sampling [Ernst et al. 2006], as we have found this gives us higher overall efficiency. The adaptive sampler is seeded with a low initial number of uniformly sampled paths per pixel, usually 32. The user also sets a target quality measure, which is internally converted to a target sample standard error. When the initial number of samples have been drawn, variance and standard error are estimated across the entire image and a sensor importance map is computed, which defines the image space sampling density for a subsequent sample budget in proportion to the difference between the pixel's estimated standard error and the user specified target. Pixels for which the standard error is below the target are disabled only if they do not fall within the region of influence of a live pixel's reconstruction filter. This process is repeated until the variance estimate for all pixels in the image fall below the target standard error.

We made the decision to track sample variance exclusively and not rely on cross-pixel estimates of the variance of the mean. While the latter measure also informs about the level of noise in the image, it is our belief that the renderer should concern itself with resolving only the noise that is produced by its own processes and leave image space denoising to be performed as a post process by tools specialized to that domain. Also, Manuka determines sample standard deviation and error as a measure relative to the local mean and not in absolute terms, which is important as noise is perceptible mostly in its magnitude relative to the signal that carries it. When doing this, care must be taken: in relative terms, noise riding on an arbitrarily dark region of the image may be quite large while also being imperceptible; furthermore, very bright regions of the image can have noise that is not visible if it is beyond the point where the image is clipped to white. Users can therefore specify the luminance range Manuka should concentrate its sampling efforts on, and samples that fall beyond this range have their relative variance smoothly dampened to avoid over sampling.

An interesting observation is that while high frequency noise is generally the most visible to humans, Monte Carlo processes actually produce noise in all frequencies (that is, the noise signal produced actually has a rather white spectrum); indeed, we observed that this low-frequency noise component became visible in many renders once users set the quality target high enough, as the early implementation of the adaptive sampler tracked pixels independently form one another. With that in mind, we added estimates of variance at various image resolutions by combining the per-pixel statistics. The final variance estimate for a given pixel is the result of estimating both its own local sample variance, as well as the variance derived from aggregating the statistics for all pixels within circles of radius from 1 to 32 pixels centered around it. While this is an expensive operation in terms of compute resources, we have found this results in more stable end results and is effective at reducing the amount of low-frequency noise left in the renders produced by Manuka.

4 PRODUCTION ESSENTIALS

Sanitising input. To enable maximum artistic creativity, we implement a few mechanisms that preprocess our input to improve performance. The reasoning behind this is that we would like to let the computer deal with technical details that may not be obvious to the user wherever possible. This includes for example the elimination of BSDF layers which are hidden under other opaque layers, deduplication of per-lobe normals. These sanitisation steps sometimes result in tremendous memory savings.

Supporting workflows. As well as being a physically-based light transport simulator, Manuka supports side channel data and back doors to facilitate established workflows in movie production. This includes the output of *arbitrary output variables* (AOVs), and concepts such as holdouts, shadow casters, optional visibility to camera, and gobos. Path length can be controlled as a user parameter, and optionally the material interface supports increasing BSDF lobe roughness with increasing number of bounces from the camera. The most extreme back door is the evaluation of traditional RSL `illuminance` loops in the shader (which is the only construct in Manuka that supports lights-as-code). Note that some of these tricks

make an implementation of a consistent bidirectional estimator hard or impossible.

Light path selection. We do not support generic light path regular expressions. Manuka has, however, builtin support for a fixed set of selectors which are tailored exactly to the needs of our users. The resulting buffers will then be combined by the compositing department, making use of them in combination with AOVs and deep buffers.

Checkpoints and restarts. We support interrupting the render and restarting it at a later point. This obviously includes the frame buffer and (quasi-)random seeds, but we also avoid the need for repeated tessellation and shading by using data caches.

5 RESULTS

We present typical numbers for the performance of our system in form of aggregate numbers for the whole facility, averaged over three weeks and all shows. Additionally, we picked three frames as example of occasionally appearing extraordinary production requirements, to demonstrate behaviour of the system in more extreme cases, see Fig. 6, Fig. 8 and Fig. 9.

Occlusion culling. We did not address occlusion culling so far in our discussion. *Shade before hit* implies that we would waste time shading vertices that would not contribute to the final render. We were concerned about this in the design phase of Manuka, and we implemented a so called *oracle pass*, which runs a coarse ray tracing pass on un-diced, un-shaded, and un-displaced input geometry, annotated by a very rough material description to coarsely mimic the ray distribution of the final render. Visibility discovered during this pass is stored, and we construct a conservative estimate of hit density that is used to inform final shading rate or even completely discard the geometry.

In the original REYES paper, the authors state that usually *Computer graphics models are like movie sets* and only model the important parts of the scene. This still seems to be true to some extent, since the oracle pass is a feature that is rarely used in practice at Weta Digital. The time it takes to do a second pass of scene ingestion often outweighs the benefits of occlusion culling. This is illustrated in Table 4: occlusion culling can bring our shading reuse from the figure *shading reuse* in to what is indicated as *delayed shading reuse*. This is different from *ideal shading reuse* in that it always shades a whole grid at once. Note how even in the complex scene from Fig. 9 where the majority of the shaded vertices are never touched for the final render, Manuka's architecture still gains a 4× speedup over shade on hit.

Shade order. Table 1 shows the effect of grid scheduling during shading. In *random order*, the shapes in the input RIB stream are shuffled before shading. This leads to incoherent access patterns to both RAM and disk storage. In *identifier* order, we sort elements in the input stream lexically according to globally unique identifier strings, which, much like file paths, match the hierarchical structure of the scene graph. As arbitrary as it may sound, this order works well for us because the naming scheme currently in use at Weta Digital generates identifiers which correlate well with the texture sets used by the corresponding assets, effectively maximizing the probability that assets that share the same textures be processed in close succession.

cache misses	Apes3 Fig. 6	Valerian Fig. 8	GOTG2 Fig. 9
shading random order	34.6M	1.5M	7.7M
shading identifier order	28.0M	0.7M	5.6M
ratio	0.81	0.47	0.72

Table 1. The effect of grid ordering on shading performance. For *random* order, the incoming shapes were randomly shuffled, while for *identifier* order, the grids were sorted by a hierarchical identifier name. These tests use 1GB of memory texture cache and were run at 2k resolution with shading rate 1. Results are given as the median over multiple runs on the same machine.

texture reduction	Apes3 Fig. 6	Valerian Fig. 8	GOTG2 Fig. 9
total textures referenced	2.1 TB	950 GB	5.5 TB
textures read from disk	105.4 GB	3.6 GB	77.6 GB
number of unique tex. files	27k	26k	22k
avg. number of layers	5.38	1.94	1.58
avg. per-vertex BSDF inputs	35.91 B	11.95 B	23.25 B
total per-vertex data	12.85 GB	1.25 GB	24.4 GB
% of original data stored	12.2%	34.7%	31.4%

Table 2. Texture size figures for the three test scenes. Manuka's per-vertex BSDF inputs facilitate compression of the input textures. The number of layers reported here is the number of layers stored during light transport, i.e. where blending has a directional dependency.

timing breakdown	Apes3 Fig. 6	Valerian Fig. 8	GOTG2 Fig. 9
shading (frontend)	1h 55m	15m	4h 28m
light transport (backend)	5h 18m	55m	4h 35m

Table 3. This table shows a coarse breakdown into shading and light transport time. The shading phase is dominated by texturing and takes up a significant percentage of the total render time.

Size of BSDF inputs. In addition to geometry, we store per-vertex BSDF inputs. We argue that this is a more compact data layout than storing textures, since these values are pre-filtered and stored at exactly the shading rate. Also, we compress and quantise the data, using domain knowledge: layer blend weights use only eight bits and are used to mask out invisible layers which are not stored at all. The smallest possible size for a simple RGB colour is 4 bytes per vertex. Using up to 14 layers of BSDF with per-lobe shading normals, we have seen BSDF inputs up to 200 bytes. A more typical BSDF might have maybe six or seven layers and with a per-vertex storage requirement in the vicinity of 100 bytes. When possible, we take advantage of storing dictionaries per grid, reducing net storage cost per vertex further. A numerical analysis of the ratio between the input textures and the output BSDF data can be found in Table 2.

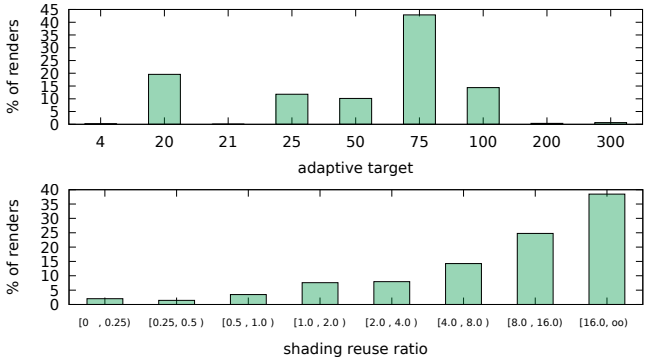

Fig. 10. Top: distribution of adaptive targets (100% would be considered a converged render) throughout the facility in the last three weeks. This is a random snapshot and numbers will vary a bit when shows are closer to delivery. For this snapshot, we get an average shading reuse ratio of 13.5×. We reach higher reuse ratios for longer running renders, which means we cut down on render times more, the more complex the render is. Bottom: well over 90% of all renders (including test renders and those with small adaptive target) profit from shading reuse, i.e. have a reuse ratio > 1×.

	Apes3	Valerian	GOTG2
shading reuse	Fig. 6	Fig. 8	Fig. 9
approx. deinstanced # vertices	23.2G	49.3G	182.5G
total #vertices	376M	75M	1.12G
#v read ≥ 1×	108M	30M	243M
total #material queries	7.3G	1.1G	4.9G
shading reuse	19.33×	14.2×	4.3×
delayed shading reuse	26.57×	17.2×	4.8×
ideal shading reuse	67.43×	35.0×	19.9×
break even # progressions	24	6.5	114
progressions rendered	466	92	493

Table 4. Analysis of actual and theoretically possible shading reuse. *Shading reuse* is what our system leverages. The *ideal shading reuse* figure is the maximum theoretically possible shading reuse if we employed shade on hit together with a perfect cache. *Delayed shading reuse* indicates how much reuse could be reached if the shading and caching happens on a per-grid basis instead of on a per-vertex basis. In the Apes3 shot, shading reuse is encumbered by sub sampling vertices in overly detailed grids (coming from the moss), not from the lack of occlusion culling.

Render time breakdown. There is some variance in how we spend the total render time budget. Generally, there is a surprisingly small share spent in ray tracing (about 20% of the render time). Shading typically takes 5% – 10% of the total runtime, and is in turn usually dominated by texture accesses (often well above 50% of the shading run time). Some breakdown numbers can be seen in Table 3. Evaluating and sampling light sources in the light hierarchy (up to 25%), as well as CDF construction for volumes are expensive, and even the run-time cost of evaluating layered BSDF is significant. As mentioned before, in terms of time to lower variance, these more involved sampling strategies proved to be useful.

Shading reuse. We quantify the amount of shading reuse that our shade before hit architecture facilitates by analysing the number of shaded vertices and comparing this to the number of material instances created during the light transport stage.

From a back-of-the-envelope computation we expect for a typical render about 2 million paths per progression[11] (for a 2k render), with a typical average of 5 bounces, and expect a converged image after 500 progressions. This results in just about five billion paths. On a typical production scene we often see about 200 million shaded vertices, which would yield about 25 times more material accesses during light transport than vertices shaded.

Indeed, when collecting statistics from three weeks worth of renders in the facility, we observe a ratio of 32.6 : 1 traced vs. shaded vertices. This number is considering converged renders, i.e. the adaptive sampling was set to reach a target of 100% for all pixels. If we include intermediate test renders which are set to accept more variance in favour of faster turnaround times, this ratio drops to about 13.5 : 1. Detailed histograms of the data can be seen in Fig. 10. None of these renders use the oracle pre-pass for occlusion culling. For individual breakdowns for the three frames, refer to *shading reuse* in Table 4. Please note that these numbers correlate the number of shaded vertices to the *number of material queries*, not to the number of vertex data reads. The difference is that constructing a material for a micropolygon would usually read four vertices and thus make the reuse number grow by about 4×. A shade on hit architecture would however shade only one point and not all four corners and then interpolate, so we deemed this comparison more useful.

Another interesting number in Table 4 is the break even number of progressions: assuming no adaptive sampling takes place, this is the number of samples per pixel we need to compute before we shade less than an ideal, hypothetical shade on hit architecture would. We compute this figure by assuming the number of material queries is equally distributed between progressions while keeping the number of shaded vertices fixed.

Note that these numbers do not quantify the advantages of more coherent texture access and potential benefits of using wider SIMD during batch shading of grids, which would be a lot harder to leverage during light transport.

6 DISCUSSION

We follow the classic REYES architecture closely and perform shading before sampling. An important difference is that shading in our system does not include BSDF evaluation. This means that we can do motion blur correctly, since shading results in BSDF inputs which are moved along the time dimension. We follow the shade-once paradigm and only compute BSDF inputs at shutter open, which may lead to slight inaccuracies for highly time-dependent shading. This occurs for instance for strong motion blur, due to changing texture filter regions. High-quality fire, sparks and rain require to make particles appear with sub shutter interval accuracy. The workaround

[11] The most used path scheduling sequence in practice at *Weta Digital* traces one path per pixel through all pixels in what we ended up calling *progressions*. This sequencing is useful for interactivity, as all of the image at any point is kept at about a uniform level of noise. It is also useful for our adaptive sampling engine, which is continuously fed data for the whole frame buffer.

used in production is to render lines with spatial shading instead of points with time-varying shading.

We also use micropolygons and exploit texture locality and vectorisation during shading. In analogy to the original paper, our architecture is designed for rendering exceedingly complex models. We mean both geometry and RSL shaders with textures, as well as complicated light transport, touching surface points with many rays, reusing shading information efficiently.

Disadvantages. Look development workflows depend on fast iteration cycles when tuning material appearance. However, our architecture is optimised for very complex models and time to final frame. We have a special mode to support this workflow, but it is much less efficient than the batch shading code path. A specialised GPU program evaluating the same shaders would be a better way of supporting look development workflows, much like Pixar's *Flow* tool [Nahmias and Pesare 2016].

Assumptions and design choices. Our architecture is designed to be fast for offline batch rendering of assets with maximum realism when it comes to geometrical details and shading complexity. In particular, faster time-to-first-image may be achieved by using a more classic shade on hit architecture. Application domains with simpler geometry and simpler shaders, such as feature animation as opposed to VFX, may benefit from moving compute efforts from more complex path space sampling techniques to post processing in the form of denoising, which usually works increasingly better for smoother surfaces with less detail. If fewer indirect bounces or more approximate indirect lighting is acceptable, an on-demand tessellation scheme, switching to coarse level of detail quickly, may be beneficial. Also, if colour accuracy is expendable but extremely bright and saturated colours are needed to achieve a specific look (again, as it might be the case for feature animation work), using RGB transport might be a valid design choice. This is a consequence of energy conservation, which dictates a limit on how bright and saturated a colour can be at the same time, when following physically-based spectral transport.

In-house vs. commercial. Manuka is an in-house product and this means we don't have to support external customers and their workflows, in theory enabling us to phase out legacy code if nobody is using it. In practice this rarely happens or takes a long time: part of the reason is that Manuka's development is very agile, the releases happen relatively frequently and are immediately adopted by some shows, while others continue using older functionality.

The main benefit of developing a renderer in-house is that we are able to quickly respond to production needs. In general, Manuka's development is more driven by user needs than by the developers. For instance the decision to not use the occlusion culling oracle is entirely a result of the kind of input our users create, while the developers were concerned about possible inefficiencies without it.

7 FUTURE CHALLENGES

Path tracing has brought simple unified workflows to production rendering. Many, if not all lighting effects can now be solved in a single-pass beauty render. In the future, we would like to further unify workflows: the push to speed up rendering may lead to extended use of the path tracing solution in fields such as layout, animation, or live pre-visualisation of motion capture on stage. This is currently the realm of specialised real-time rendering software, requiring specifically simplified input geometry and material descriptions. Merging these scene representations and rendering algorithms seems like a worthwhile effort.

But even within the comparatively tight specifications of a renderer for final frames, there is room to improve the *time to first iteration*. Note that this is different to *time to first pixel*. Artists require a reasonably converged image to be able to judge the look. Optimising for this requires to address speed issues in the whole pipeline, carrying data all the way from the user input to the displayed frame. Much can be gained by simplifying the parameter space which has to be explored when fine tuning light transport algorithms, for instance by just providing one monolithic, complex algorithm that always works. On the other hand, we have a good repertoire of specialised solutions, such as for rendering hair, or caustics with manifold next event estimation, which greatly outperform general solutions in applicable cases.

General approaches to light transport (for instance VCM or gradient domain path tracing) can be very expensive and exhibit problems in specific cases. For instance the photon density estimation using a disk area or a sphere volume both fail in hair. Gradient domain path tracing produces artifacts if the underlying sampler used to compute gradients is too noisy. These spurious fail cases combined with significant overhead are the reason why any new algorithm has to answer the tough question: "can we turn it on by default?".

To that end it would be a great step to be able to automatically educate users about inefficiencies in the renderer caused by a particular scene setup. For instance it is obvious to an expert developer that modelling a solid glass coating around an object is clearly less efficient than using a multi-layer BSDF.

ACKNOWLEDGMENTS

We share a powerful memory of the early days on Manuka: our project involved a large number of people besides the authors and we would like to start acknowledging that this body of work was made possible thanks to Joe Letteri and Sebastian Sylwan: they entrusted us with the mission, gave us the time and resources to make it happen and provided us with a great richness of inspiration and courage to stay true to our path, to push boundaries and to keep moving forwards. We are especially grateful for the daily discussions with Joe providing access to a wealth of information in terms of background and inspiration, as well as insight into the process of moviemaking and how it evolved during his career. Being able to unroll the history of a number of commonly accepted industry practices gave us key insights during the design of the architecture, and helped us understand what features we needed to keep and what instead were better reconsidered or redesigned from the ground up. After *Manuka* became the chosen production renderer at Weta Digital, many users from the look development, lighting and effects disciplines have also contributed valuable insights and thought-provoking observations, and we're sure we wouldn't have as good a system now without their help. We would also like to thank the many members of the Manuka team over the years: Jiří Vorba, Shijun Haw, Kimball Thurston, Peter Pearson, Tom Matterson, Christian

Hipp, as well as Leonhard Grünschloß, Eugene d'Eon, Ralf Karrenberg, Sehera Nawaz, Liana Fleming, Daniel Lond, Jan Althaus, Patrick Kelly, Robin Hub, Antoine Bouthors, Derek Gerstmann. No system of this size would have any hope of success without good documentation, for which Carla Morris was key, and proper testing and validation, thanks to Björn Siegert and Arthur Terzis. In the end, many visiting researchers have enriched the project bringing in new ideas and inspirations to our collective work: Alexander Wilkie, Wenzel Jakob, Jaroslav Křivánek, Florian Simon, Iliyan Georgiev, Marco Manzi, Markus Kettunen, Tzu-Mao Li, Christopher Corsi, Jordan Gestring, Jeff Stuart, Javor Kalojanov, Ling-Qi Yan.

REFERENCES

Steve Agland. 2014. CG Rendering and ACES. http://nbviewer.ipython.org/gist/sagland/3c791e79353673fd24fa. (2014).

Timo Aila, Tero Karras, and Samuli Laine. 2013. On Quality Metrics of Bounding Volume Hierarchies. In *Proceedings of High Performance Graphics (HPG '13)*. 101–107. https://doi.org/10.1145/2492045.2492056

Carsten Benthin, Sven Woop, Ingo Wald, and Attila T. Áfra. 2017. Improved Two-level BVHs Using Partial Re-braiding. In *Proceedings of High Performance Graphics (HPG '17)*. Article 7, 8 pages. https://doi.org/10.1145/3105762.3105776

Brian Budge, Tony Bernardin, Jeff Stuart, Shubhabrata Sengupta, Kenneth Joy, and John Owens. 2009. Out-of-core data management for path tracing on hybrid resources. In *Computer Graphics Forum (Proc. of Eurographics)*. 385–396.

Brent Burley and Dylan Lacewell. 2008. Ptex: Per-face Texture Mapping for Production Rendering. In *Proceedings of the Nineteenth Eurographics Conference on Rendering (EGSR '08)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 1155–1164. https://doi.org/10.1111/j.1467-8659.2008.01253.x

Nick Chirkov. 2005. Fast 3D Line Segment–Triangle Intersection Test. *Journal of Graphics, Gpu, and Game Tools* 10, 3 (2005), 13–18.

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

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

Per Christensen and Wojciech Jarosz. 2016. The Path to Path-Traced Movies. *Foundations and Trends in Computer Graphics and Vision* 10 (2016), 103–175. Issue 2.

CIE. 1996. *The Basis of Physical Photometry.* Commission Internationale de l'Eclairage.

CIE. 2004. *Colorimetry.* Technical Report. Commission Internationale de l'Eclairage.

David Cline, Justin Talbot, and Parris K. Egbert. 2005. Energy Redistribution Path Tracing. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 24, 3 (2005), 1186–1195.

Robert Cook, Loren Carpenter, and Edwin Catmull. 1987. The Reyes Image Rendering Architecture. *Computer Graphics (Proc. SIGGRAPH)* 21, 4 (1987), 95–102.

Dennis Couzin. 2007. Optimal fluorescent colors. *Color Research & Application* 32, 2 (2007), 85–91.

R. R. Coveyou, V. R. Cain, and K. J. Yost. 1967. Adjoint and Importance in Monte Carlo Application. *Nuclear Science and Engineering* 27, 2 (1967), 219–234. https://doi.org/10.13182/NSE67-A18262

Christopher DeCoro, Tim Weyrich, and Szymon Rusinkiewicz. 2010. Density-based Outlier Rejection in Monte Carlo Rendering. *Computer Graphics Forum (Proc. Pacific Graphics)* 29, 7 (Sept. 2010), 2119–2125.

Eugene D'Eon and Geoffrey Irving. 2011. A Quantized-diffusion Model for Rendering Translucent Materials. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 30, 4, Article 56 (July 2011), 14 pages.

Michael Donikian, Bruce Walter, Kavita Bala, Sebastian Fernandez, and Donald P. Greenberg. 2006. Accurate Direct Illumination Using Iterative Adaptive Sampling. *IEEE Transactions on Visualization and Computer Graphics* 12, 3 (May 2006), 353–364.

Christian Eisenacher, Gregory Nichols, Andrew Selle, and Brent Burley. 2013. Sorted Deferred Shading for Production Path Tracing. *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering)* 32, 4 (2013), 125–132.

Manfred Ernst, Marc Stamminger, and Günther Greiner. 2006. Filter Importance Sampling. In *Proc. IEEE Symposium on Interactive Ray Tracing*. 125–132.

Hugh Fairman, Michael Brill, and Henry Hemmendinger. 1998. How the CIE 1931 color-matching functions were derived from Wright-Guild data. *Color Research and Application* 22, 1 (1998), 11–23.

Luca Fascione, Johannes Hanika, Marcos Fajardo, Per Christensen, Brent Burley, and Brian Green. 2017a. Path Tracing in Production – Part 1: Writing Production Renderers. In *SIGGRAPH Courses*.

Luca Fascione, Johannes Hanika, Rob Pieké, Christopher Kulla, Christophe Hery, Ryusuke Villemin, Thorsten-Walther Schmidt, Daniel Heckenberg, and André Mazzone. 2017b. Path Tracing in Production – Part 2: Making movies. In *SIGGRAPH Courses*.

Graham D. Finlayson and Steven D. Hordley. 2000. Improving gamut mapping color constancy. *IEEE Transactions on Image Processing* 9, 10 (2000), 1774–1783.

Matthew Fisher, Kayvon Fatahalian, Solomon Boulos, Kurt Akeley, William R. Mark, and Pat Hanrahan. 2009. DiagSplit: Parallel, Crack-free, Adaptive Tessellation for Micropolygon Rendering. *ACM Trans. Graph.* 28, 5, Article 150 (Dec. 2009), 10 pages. https://doi.org/10.1145/1618452.1618496

D. A. Forsyth. 1990. A Novel Algorithm for Color Constancy. *Int. J. Comput. Vision* 5, 1 (Sept. 1990), 5–36.

Jeppe Revall Frisvad, Toshiya Hachisuka, and Thomas Kim Kjeldsen. 2014. Directional Dipole Model for Subsurface Scattering. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 34, 1, Article 5 (Dec. 2014), 12 pages.

Kirill Garanzha and Charles Loop. 2010. Fast Ray Sorting and Breadth-First Packet Traversal for GPU Ray Tracing. In *Computer Graphics Forum (Proc. of Eurographics)*. 289–298.

Iliyan Georgiev. 2012. *Implementing Vertex Connection and Merging.* Technical Report. Saarland University. http://www.iliyan.com/publications/ImplementingVCM

Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. 2012. Light Transport Simulation with Vertex Connection and Merging. *ACM Trans. on Graphics (Proc. SIGGRAPH Asia)* 31, 6 (2012), 192:1–192:10.

Andrew Glassner. 1994. A Model for Fluorescence and Phosphorescence. In *Proceedings of the 5th Eurographics Workshop on Rendering*. 57–68.

Jeffrey Goldsmith and John Salmon. 1987. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics & Applications* 7, 5 (1987), 14–20.

Larry Gritz (Ed.). 2009. *Open Shading Language.* Culver City, CA, USA. http://opensource.imageworks.com/?p=osl

Leonhard Grünschloß, Martin Stich, Sehera Nawaz, and Alexander Keller. 2011. MSBVH: An Efficient Acceleration Data Structure for Ray Traced Motion Blur. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics (HPG '11)*. 65–70.

Toshiya Hachisuka, Jacopo Pantaleoni, and Henrik Wann Jensen. 2012. A Path Space Extension for Robust Light Transport Simulation. *ACM Trans. on Graphics (Proc. SIGGRAPH Asia)* 31, 6 (2012), 191:1–191:10.

Johannes Hanika. 2011. *Spectral Light Transport Simulation using a Precision-based Ray Tracing Architecture.* Ph.D. Dissertation. Ulm University.

Johannes Hanika, Marc Droske, and Luca Fascione. 2015. Manifold Next Event Estimation. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering)* 34, 4 (June 2015), 87–97.

Johannes Hanika, Peter Hillman, Martin Hill, and Luca Fascione. 2012. Camera Space Volumetric Shadows. In *Proceedings of Digital Production Symposium*. 7–14.

Johannes Hanika, Alexander Keller, and Hendrik Lensch. 2010. Two-Level Ray Tracing with reordering for highly complex Scenes. In *Proc. of Graphics Interface 2010*. 145–152.

Jon Yngve Hardeberg, Hans Brettel, and Francis J. M. Schmitt. 1998. Spectral characterization of electronic cameras. In *Proc. SPIE*. 3409:1–3409:10. https://doi.org/10.1117/12.324101

Qiming Hou, Hao Qin, Wenyao Li, Baining Guo, and Kun Zhou. 2010. Micropolygon ray tracing with defocus and motion blur. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 29, 4 (2010), 1–10.

Image-Engineering. 2010. Camspecs Express. https://www.image-engineering.de/products/equipment/measurement-devices/588-camspecs-express. (2010).

ITU. 2002. *Recommendation ITU-R BT.709-5: Parameter values for the HDTV standards for production and international programme exchange.* Technical Report.

Thiago Ize. 2013. Robust BVH Ray Traversal. *Journal of Computer Graphics Techniques (JCGT)* 2, 2 (19 July 2013), 12–27. http://jcgt.org/published/0002/02/02/

Wenzel Jakob and Steve Marschner. 2012. Manifold exploration: a Markov chain Monte Carlo technique for rendering scenes with difficult specular transport. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 31, 4 (2012), 58:1–58:13.

Jun Jiang, Dengyu Liu, Jinwei Gu, and Sabine Susstrunk. 2013. What is the Space of Spectral Sensitivity Functions for Digital Color Cameras?. In *IEEE Workshop on the Applications of Computer Vision (WACV)*.

Andreas Karge, Jan Fröhlich, and Bernd Eberhardt. 2014. Open Film Tools – Camera Characterization for Cinematographers. https://www.hdm-stuttgart.de/open-film-tools/english/publications/OFT-CameraCharacterization.pdf. (2014).

Rei Kawakami, Zhao Hongxun, Robby T. Tan, and Katsushi Ikeuchi. 2013. Camera Spectral Sensitivity and White Balance Estimation from Sky Images. *International Journal of Computer Vision* (June 2013).

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

Alexander Keller, Luca Fascione, Marcos Fajardo, Per Christensen, Johannes Hanika, Christian Eisenacher, and Greg Nichols. 2015. The Path-Tracing Revolution in the Movie Industry. In *SIGGRAPH Courses*.

Alexander Keller, Carsten Wächter, Matthias Raab, Daniel Seibert, Dietgar van Antwerpen, J. Korndörfer, , and L. Kettner. 2017. The Iray Light Transport Simulation and Rendering System. arXiv:1705.01263 [cs.GR]. (May 2017).

Markus Kettunen, Marco Manzi, Miika Aittala, Jaakko Lehtinen, Frédo Durand, and Matthias Zwicker. 2015. Gradient-Domain Path Tracing. *ACM Trans. on Graphics*

(Proc. SIGGRAPH) 34, 4 (2015), 123:1–123:13.
Alan King, Christopher Kulla, Alejandro Conty, and Marcos Fajardo. 2013. BSSRDF Importance Sampling. In *SIGGRAPH Talks*.
Christopher Kulla and Marcos Fajardo. 2011. Importance Sampling of Area Lights in Participating Media. In *SIGGRAPH Talks*. 55:1–55:1.
Peter Kutz, Ralf Habel, Yining Karl Li, and Jan Novák. 2017. Spectral and Decomposition Tracking for Rendering Heterogeneous Volumes. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 36, 4 (July 2017), 111:1–111:16.
Jaroslav Křivánek and Eugene d'Eon. 2014. A Zero-variance-based Sampling Scheme for Monte Carlo Subsurface Scattering. In *SIGGRAPH Talks*. 66:1–66:1.
Eric Lafortune and Yves Willems. 1993. Bi-Directional Path Tracing. In *Proc. of COMPUGRAPHICS*. 145–153.
K. M. Lam. 1985. *Metamerism and Colour Constancy*. Ph.D. Dissertation. University of Bradford.
Bernd Lamparter, Heinrich Müller, and Jörg Winckler. 1990. *The Ray-z-Buffer—An Approach for Ray Tracing Arbitrarily Large Scenes*. Technical Report. Albert-Ludwigs University at Freiburg.
Jun S. Liu, Faming Liang, and Wing Hung Wong. 2000. The Multiple-Try Method and Local Optimization in Metropolis Sampling. *J. Amer. Statist. Assoc.* 95, 449 (March 2000), 121–134.
David L. MacAdam. 1935a. Maximum Visual Efficiency of Colored Materials. *Journal of the Optical Society of America)* 25, 11 (1935), 361–367.
David L. MacAdam. 1935b. The Theory of the Maximum Visual Efficiency of Colored Materials. *Journal of the Optical Society of America)* 25, 8 (1935), 249–249.
David J. MacDonald and Kellogg S. Booth. 1990. Heuristics for ray tracing using space subdivision. *The Visual Computer* 6, 3 (1990), 153–166.
Laurence T. Maloney. 1986. Evaluation of linear models of surface spectral reflectance with small numbers of parameters. *Journal of the Optical Society of America)* 3, 10 (1986), 1673–1683.
Stephen McAuley, Stephen Hill, Naty Hoffman, Yoshiharu Gotanda, Brian Smits, Brent Burley, and Adam Martinez. 2012. Practical Physically-based Shading in Film and Game Production. In *SIGGRAPH Courses*. 10:1–10:7.
Nick McKenzie, Martin Hill, and Jon Allitt. 2010. Rendering "Avatar": Spherical Harmonics in Production. (2010). SIGGRAPH Talks.
Johannes Meng, Florian Simon, Johannes Hanika, and Carsten Dachsbacher. 2015. Physically Meaningful Rendering using Tristimulus Colours. *Proc. Eurographics Symposium on Rendering* 34, 4 (June 2015), 31–40.
Gary W. Meyer. 1988. Wavelength Selection for Synthetic Image Generation. *Comput. Vision Graph. Image Process.* 41, 1 (Jan. 1988), 57–79.
Bochang Moon, Yongyoung Byun, Tae-Joon Kim, Pio Claudio, Hye-Sun Kim, Yun-Ji Ban, Seung Woo Nam, and Sung-Eui Yoon. 2010. Cache-oblivious Ray Reordering. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 29, 3 (July 2010), 28:1–28:10.
Jn Morovi. 2008. *Color Gamut Mapping*. Wiley Publishing.
Jean-Daniel Nahmias and Davide Pesare. 2016. Look Development in Real Time. (2016). Siggraph NVIDIA Presentations.
Jan Novák, Andrew Selle, and Wojciech Jarosz. 2014. Residual ratio tracking for estimating attenuation in participating media. *ACM Trans. on Graphics (Proc. SIGGRAPH Asia)* 33, 6 (Nov. 2014), 179:1–179:11. https://doi.org/10.1145/2661229.2661292
Jacopo Pantaleoni, Luca Fascione, Martin Hill, and Timo Aila. 2010. PantaRay: Fast Ray-traced Occlusion Caching of Massive Scenes. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 29, 3 (2010), 1–10.
Mark S. Peercy. 1993. Linear Color Representations for Full Speed Spectral Rendering. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*. 191–198.
Matt Pharr and Pat Hanrahan. 1996. Geometry Caching for Ray-Tracing Displacement Maps. In *Proc. Eurographics Workshop on Rendering*. 31–40.
Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2017. *Physically Based Rendering: From Theory to Implementation* (3rd ed.). Morgan Kaufmann.
Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. 1997. Rendering Complex Scenes with Memory-Coherent Ray Tracing. In *Proc. of SIGGRAPH '97*. 101–108.
Pixar Animation Studios. 2015. *RenderMan 20 documentation*.
Matthias Raab, Daniel Seibert, and Alexander Keller. 2008. Unbiased Global Illumination with Participating Media. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*. 591–606.
Shaun D. Ramsey, Kristin Potter, and Charles Hansen. 2004. Ray Bilinear Patch Intersections. *Journal of Graphics Tools* 9, 3 (2004), 41–47. https://doi.org/10.1080/10867651.2004.10504896
Charles Schmidt and Brian Budge. 2002. Simple Nested Dielectrics in Ray Traced Images. *Journal of Graphics Tools* 7, 2 (2002), 1–8.
Erwin Schrödinger. 1919. Theorie der Pigmente größter Leuchtkraft. *Annalen der Physik* 367, 15 (1919), 603–622.
Peter Shirley and Changyaw Wang. 1991. Direct Lighting Calculation by Monte Carlo Integration. In *Proc. Eurographics Workshop on Rendering*. 54–59.
Florian Simon, Johannes Hanika, Tobias Zirr, and Carsten Dachsbacher. 2017. Line Integration for Rendering Heterogeneous Emissive Volumes. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering)* 36, 4 (June 2017), 101–110.
Thomas Smith and John Guild. 1931. The C.I.E. colorimetric standards and their use. *Transactions of the Optical Society* 33, 3 (1931), 73–134.
Brian Smits. 1999. An RGB-to-spectrum conversion for reflectances. *Journal of Graphics Tools* 4, 4 (1999), 11–22.
Brian Smits, Peter Shirley, and Michael Stark. 2000. Direct Ray Tracing of Displacement Mapped Triangles. In *Proc. Eurographics Workshop on Rendering*. 307–318.
Michael Stokes, Matthew Anderson, Srinivasan Chandrasekar, and Ricardo Motta. 1996. A Standard Default Color Space for the Internet — sRGB. http://www.color.org/contrib/sRGB.html. (November 1996).
Gordon Stoll, William Mark, Peter Djeu, Rui Wang, and Ikrima Elhassan. 2006. *Razor: An architecture for dynamic multiresolution ray tracing*. Technical Report 06-21. Department of Computer Science, University of Texas at Austin.
Yinlong Sun, F. David Fracchia, Thomas W. Calvert, and Mark S. Drew. 1999. Deriving Spectra from Colors and Rendering Light Interference. *IEEE Comput. Graph. Appl.* 19, 4 (July 1999), 61–67.
T. M. Sutton, F. B. Brown, F. G. Bischoff, D. B. MacMillan, C. L. Ellis, J. T. Ward, C. T. Ballinger, D. J. Kelly, and L. Schindler. 1999. *The physical models and statistical procedures used in the RACER Monte Carlo Code*. Technical Report KAPL-4840. Knolls Atomic Power Laboratory, Niskayuna, NY, USA. https://doi.org/10.2172/767449
László Szécsi, László Szirmay-Kalos, and Csaba Kelemen. 2003. Variance Reduction for Russian Roulette. *Journal of WSCG* 11, 1 (2003), 1–8.
USAS and ASME. 1967. *USA Standard Letter Symbols for Illuminating Engineering*. United States of America Standards Institute.
Eric Veach and Leonidas Guibas. 1994. Bidirectional Estimators for Light Transport. In *Proc. Eurographics Workshop on Rendering*. 147–162.
Eric Veach and Leonidas J Guibas. 1995. Optimally combining sampling techniques for Monte Carlo rendering. *Proc. SIGGRAPH* (1995), 419–428.
Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. 2014. On-line Learning of Parametric Mixture Models for Light Transport Simulation. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 33, 4 (Aug. 2014), 101:1–101:11.
Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. 2005. Lightcuts: A Scalable Approach to Illumination. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 24, 3 (July 2005), 1098–1107.
Qiqi Wang, Haiying Xu, and Yinlong Sun. 2004. Practical construction of reflectances for spectral rendering. In *In Proceedings of the 22th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. 193–196.
Rui Wang and Oskar Åkerlund. 2009. Bidirectional Importance Sampling for Unstructured Direct Illumination. *Computer Graphics Forum* 28, 2 (2009), 269–278.
Greg Ward and Elena Eydelberg-Vileshin. 2002. Picture Perfect RGB Rendering Using Spectral Prefiltering and Sharp Color Primaries. In *Eurographics Workshop on Rendering*. The Eurographics Association, 117–124.
Andrea Weidlich and Alexander Wilkie. 2009. Rendering the effect of labradorescence. In *Graphics Interface*. 79–85.
Alexander Wilkie, Sehera Nawaz, Marc Droske, Andrea Weidlich, and Johannes Hanika. 2014. Hero Wavelength Spectral Sampling. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering)* 33, 4 (July 2014), 123–131.
Andrew Woo, Andrew Pearce, and Marc Ouellette. 1996. It's Really Not a Rendering Bug, You See ... *IEEE Comput. Graph. Appl.* 16, 5 (Sept. 1996), 21–25.
E. R. Woodcock, T. Murphy, P. J. Hemmings, and T. C. Longworth. 1965. 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.
Magnus Wrenninge. 2016. Efficient rendering of volumetric motion blur using temporally unstructured volumes. *Journal of Computer Graphics Techniques* 5, 1 (2016), 1–34.
William David Wright. 1928. A re-determination of the trichromatic coefficients of the spectral colours. *Transactions of the Optical Society* 30, 4 (1928), 141–164.
G. Wyszecki and W. S. Stiles. 2000. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. John Wiley & Sons.
Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and Sung-Eui Yoon. 2015. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. *Computer Graphics Forum (Proceedings of Eurographics)* 34, 2 (May 2015), 667–681. https://doi.org/10.1111/cgf.12592