Identifying and Visualizing Computational Hotspots in Path Tracing

Mathanrajan Sundarrajan
Supervisor(s): Mark van de Ruit, Elmar Eisemann
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Identifying and Visualizing Computational Hotspots in Path Tracing

M. Sundarrajan, M. van de Ruit, and E. Eisemann

Delft University of Technology, The Netherlands

Abstract

Path tracing is a well-known light transport algorithm used to render photo-realistic images. However, it is an expensive algorithm with an active area of research for improving its efficiency. In our work, we present a method to measure and visualize the regions of high computational cost for unidirectional path tracers. We have defined metrics to estimate the cost per pixel that can be visualized as a two-dimensional(2D) image. To show its usefulness, we present how our method can visualize the effect of changing material properties, object placement and other scene variables on computational cost. With this insight, the user can make clever choices to improve computational time.

1. Introduction

Path tracing is a rendering technique used to produce photo-realistic images. This technique falls under the category of light transport algorithms and these algorithms have become the standard for producing realistic images in the VFX industry. However, this is a computationally heavy technique. Even with the best machines, it can take hours and even days to render a scene. This defines the need to improve computational time while maintaining photo-realistic properties.

There is a significant body of research in improving rendering hardware and light transport algorithms such as path tracing [DNL*17, WMG*09]. Along with these improvements, involving the users can provide additional impact on computational time. A method that provides the user, namely artists, insight into how their designs affect computational time could empower the artist to make clever scene changes. These scene changes can reduce computational costs and potentially negate weaknesses in the rendering engine.

For this purpose, we propose metrics for the estimating the computational costs per pixel in a rendered image, which reflects the contributions of changes an artist can make for unidirectional path tracers. These include changes in material properties, object placement, and other scene variables. These metrics can be used to visualize the costs per scene and compare costs between scenes to showcase the effect of scenic changes on computational costs.

We first cover relevant related work in Section 2. We then explore the different components of our method in Section 3. Next, we discuss details of our implementation in Section 4, which we evaluate for a variety of scenarios in Section 5. Afterwards, in Section 6, we cover the responsibility of our research. Finally, we discuss our method in Section 7 before concluding in Section 8.

2. Background and Related Work

Unidirectional path tracers

Path tracing is a light transport algorithm that faithfully simulates the true nature of light to produce an unbiased photo render of a scene. It uses Monte Carlo methods [AK90] to solve the rendering equation proposed by Kajiya [Kaj86]. In its simplest form, the algorithm traces a significant amount of paths that light takes in a scene. Since only the light that hits the camera is seen in an image, the light paths are traced from the camera to the light source in naïve unidirectional path tracers.

In the case of a unidirectional path tracer, a path can be defined as a sequence of light rays, each cast from where the previous ray intersected in the scene, starting from the camera. We find where a ray intersects by performing intersection tests against the scene. Scenes are represented as a collection of shapes, typically triangles [PJH16]. However, complex scenes can easily have hundreds of thousands of triangles [Lum17, NHB17]. Performing intersection tests on

all those triangles is very expensive. In order to narrow down the number of intersection tests needed, acceleration structures are used [HM08].

After the intersection point is found, the path can terminate or cast new rays. The cause of termination can be due to the maximum allowed path length, Russian roulette (RR) [AK90], or zero-energy carried in a path. However, it can vary between rendering engines. If new rays are to be cast, then depending on the material at the intersection point, calculations are performed to evaluate the direction of the newly cast ray spawned from the intersection point.

The above provides a significantly summarized overview of the unidirectional path tracer. We highly recommend looking into PBR book [PJH16] for a further in-depth explanation.

Evaluating render time costs

Rendering scenes with a path tracer is expensive due to various factors. To estimate the costs, analysis and heuristics metrics can be defined. The work of Wimmer *et al*. [WW03] shows how rendering time can be estimated for real-time rendering. Additionally, they proposed heuristics for measuring the costs of rendering. As paths are defined as a sequence of recursively cast rays in its simplest form, looking into the costs of a ray provides insight into the computational complexity [RKJ96, WS95, ABCC02]. From these analyses, we can see the importance of an acceleration structure concerning the cost.

Visualizing path tracing

Applying visualization techniques to path tracing enables the users to gain a better insight into path tracing. Simons *et al.* [SHP*18] previously showed how visual analytics could be used with path tracing to optimize rendering and perform further analysis. It can also aid in finding the causes of artefacts. Furthermore, Interactive visualizations can further aid in debugging, educational and performance analysis of path tracing [GFE*12].

3. Methodology

We now detail our approach to providing visual support tooling for artists. In this respect, our approach highlights regions of potentially high computational cost for a unidirectional path tracer. We also show how the computational costs are affected by changing scene variables such as material properties. To address this problem, we first introduce the notion of a computational hotspot and define how to describe it using a metric. Afterward, we describe how this metric can be measured efficiently inside a unidirectional path tracer. Finally, we detail approaches to visualizing and comparing measurements of this metric.

Computational hotspot and metrics

Computational hotspots are regions where computational costs are higher than the average cost of all the pixels in an image. To spot these regions, we first look at where a ray spends most of its computational time. In our observations profiling the PBRT-v3 renderer, we find that most computational time is spent on ray traversal and visibility computations. This matches the findings of Vasiou *et al*. [VSM*18].

Following our observation from the profiler, the major contributor to the high time consumption is dependent on the intersection tests. The actual costs for these tests highly depend on the type of acceleration structure used to store these shapes. Reinhard *et al*. and Walter *et al*. provided an in-depth analysis of the costs of these structures [RKJ96, WS95]. In general, the cost can be defined as the following heuristic:

$$C(r) = c_0 N_{\text{trav}} + c_1 N_{\text{shapes}} \tag{1}$$

Here r is a ray, c_0 and c_1 are constants dependent on the acceleration structure, N_{trav} is the number of traversals taken by the ray r within the acceleration structure, and N_{shapes} is the number of shapes for which intersection tests were taken with the ray r.

An important thing to note here is that shooting new rays always adds to the cost. Thus, it can be used as a metric to evaluate computational costs. The main advantage of having path length as a metric is that it is relatively easy to integrate onto existing rendering engines. However, it does not represent the true cost of rendering a scene. Defining the path length metric similar to Equation 1 would be $C_{\text{ray}} = 1$.

Gathering the data

The metrics mentioned above are defined as a cost function of a ray. To get the cost of the path, the costs of all the rays cast in a path are accumulated. The cost of a ray can further be split into two separate costs. One for the cost of tracing the ray, and the other for tracing the shadow ray. Note that a naïve unidirectional path tracer does not cast any shadow ray. So in cases where a shadow ray is not cast, the cost function for those rays will be 0.

This leads us to a general cost function of a path as follows:

$$C(p) = \sum_{i=0}^{N-1} \left(C_{\text{tr}}(r_i) + C_{\text{vis}}(r_i)\right) \tag{2}$$

Here a path p is defined as a set consisting of N rays such that $p = \{r_0, ..., r_{N-1}\}$, C_{tr} is the cost of tracing a ray to it's intersection point and C_{vis} is the cost of performing visibility computations on the ray.

To gather this cost, we need to trace a significant amount of paths throughout the scene. There are different approaches to this. For example, we can trace paths from the light source and then connect the paths with the camera using an approach such as photon mapping [Jen01]. To keep

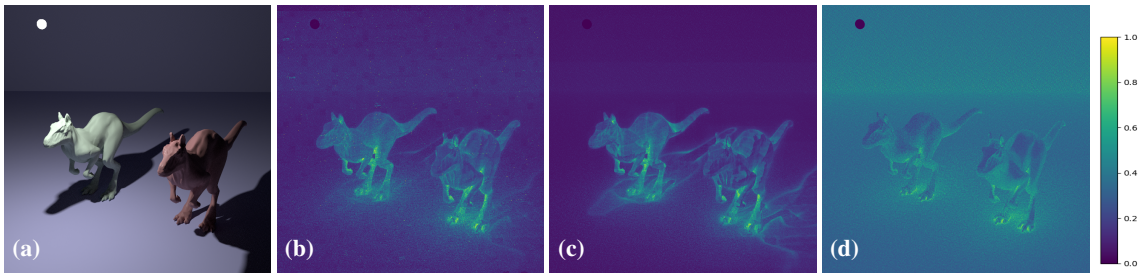


Figure 1: (a) True image of the scene and heat map visualizations of (b) time taken, (c) intersection costs and (d) path length metrics. For each heatmap (b), (c), (d), the values are normalized between 0 and 1.

the approach simple, we trace paths from the camera like a generic unidirectional path tracer.

An accumulator collects the data for each path traced from a pixel. To get the expected cost of rendering a pixel, the accumulated costs are averaged based on the number of sample paths taken per pixel. This cost per pixel is then exported for visualization and analysis.

Visualizing results

As the data is calculated per pixel, we represent the data similar to a rendered 2-dimensional image. The data is visualized by applying color maps to showcase the costs effectively. However, outliers can potentially affect the range of colors shown. To overcome this, the data distribution needs to be analyzed to determine the range of values that needs to be visualized. This step does not always have to be taken as these outlier values can indicate the problematic areas that need to be investigated. Contemporary outlier removal and noise reduction algorithms can be applied to the data before visualization.

4. Implementation

We implemented our method on top of PBRT v3 [PJH16] as an extension of the existing *PathIntegrator* structure, which passes custom data structures encapsulating said metric data.

Note that additional functions must be modified to pass the metric data depending on the data collected. In the case of the intersection costs metric (Equation 1), the function performing intersection tests with acceleration structure and shapes, estimating direct lighting and other intermediate function calls had to be modified. However, the function modifications mentioned earlier are unnecessary for collecting the path length metric. Additionally, as the path length has a fixed cost for all shadow rays, those rays need not be traced while collecting this metric. This can lead to a significant speedup in accumulating the metrics as tracing those rays have a high computational cost.

After the cost of a path sample is accumulated, it is added to the pixel cost. This process is identical to how an integrator accumulates radiance, except that the *Spectrum* value received from tracing the path contains the metric data in the color channels instead with a uniform path weight of 1. This way, the pixel cost is an unweighted average of the path costs.

5. Evaluation

We evaluate our metrics by comparing them against the ground truth represented by measuring the time spent calculating the outgoing pixel's radiance in microseconds. As thread and process scheduling introduces inaccuracies, the time metric is considerably noisy. To reduce the impact on the noise on the color maps, the data is despiked using a 3x3 median filter on values higher than the 99.9th percentile. All the test scenes uses the BVH acceleration structure and the constants $c_0 = 1$ and $c_1 = 1$ for the intersection costs (Equation 1).

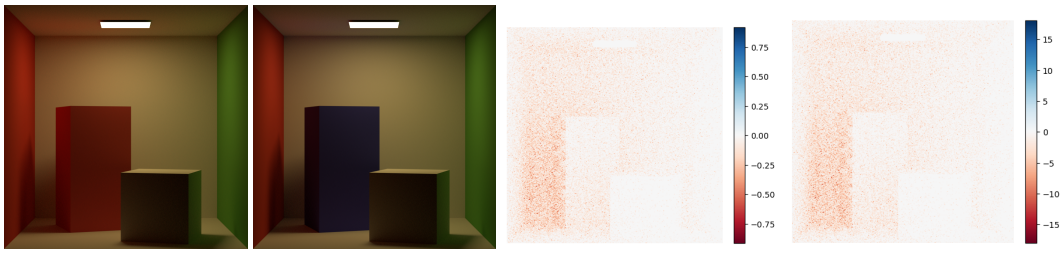
The metric is taken from its respective channel to be visualized by applying a color map. Unless specified in the test scenes, the data is not modified. If the data has few extreme outliers, it is clipped depending on the data distribution of the absolute values. To compare and visualize the metric costs between different scenes, red-blue diverging color map is applied to the cost difference.

To evaluate our method, we first compare our metrics' performance to the time taken to render a pixel. Next, we show how our method can be used to see the effect of scene changes by comparing costs. Finally, we show a simple example where a problematic region can be spotted and addressed using our method.

Correspondence with the time taken

For our first test scene, we look into the killeroo scene provided by PBRT [PJH16] in Figure 1. The actual image is rendered with 128 spp. and the metrics with 10 spp. All the images are rendered with a maximum path depth of 15.

Looking at Figure 1, we can see that the intersection met-



(a) Red box (b) Blue box (c) Path length difference (d) Intersection cost difference

Figure 2: Cornell box rendered with (a) red matte material and (b) blue matte material. The cost difference in rendering the (b) compared to (a) is shown for (c) path length and (d) intersection costs. We can see how the estimated costs are lowered for (b) the blue box scene. This is due to low throughput and Russian roulette.

ric corresponds well with the time taken as the region it highlights closely matches that of the time metric. On the other hand, the path length is not as accurate as the intersection metric but still highlights the hotspots caused due to paths stuck in the geometry.

Comparing scene costs

We demonstrate an application of our technique by comparing the cost differences of similar scenes to pick the least computationally expensive. To showcase this, we will use a modified version of the Cornell box with two different color schemes. In these scenes, the tall matte-material box is either colored in red ($Kd = \{0.5, 0.1, 0.1\}$) or blue ($Kd = \{0.1, 0.1, 0.5\}$). Given these two scenes, the costs for rendering the scene through the same camera properties are compared to evaluate the cost differences across the scenes.

We rendered the scenes as mentioned earlier with 128 spp. and the metrics with 8 spp. as shown in Figure 2. All the renders were rendered with a maximum allowed path depth of 15. The absolute costs after the 99.9th percentile are clipped.

Looking at the results shown in Figure 2, we can see that both the intersection metric and the path length metric have similar changes. This is likely because only changing an object's color would lead to a potential increase/decrease in some path's throughput but not their trajectory. As in the PBRT renderer, path throughput is considered to terminate the path either due to zero throughput or Russian roulette. The probability of terminating a path through Russian roulette is dependent on the throughput of the highest contributing color channel. Where high throughput corresponds to a lower chance of termination due to RR, the effects of changes in path throughput can be directly seen through the path length metric. Evidently, in the Cornell box scene, costs contributed by intersection tests and the acceleration structure are relatively uniform throughout the scene.

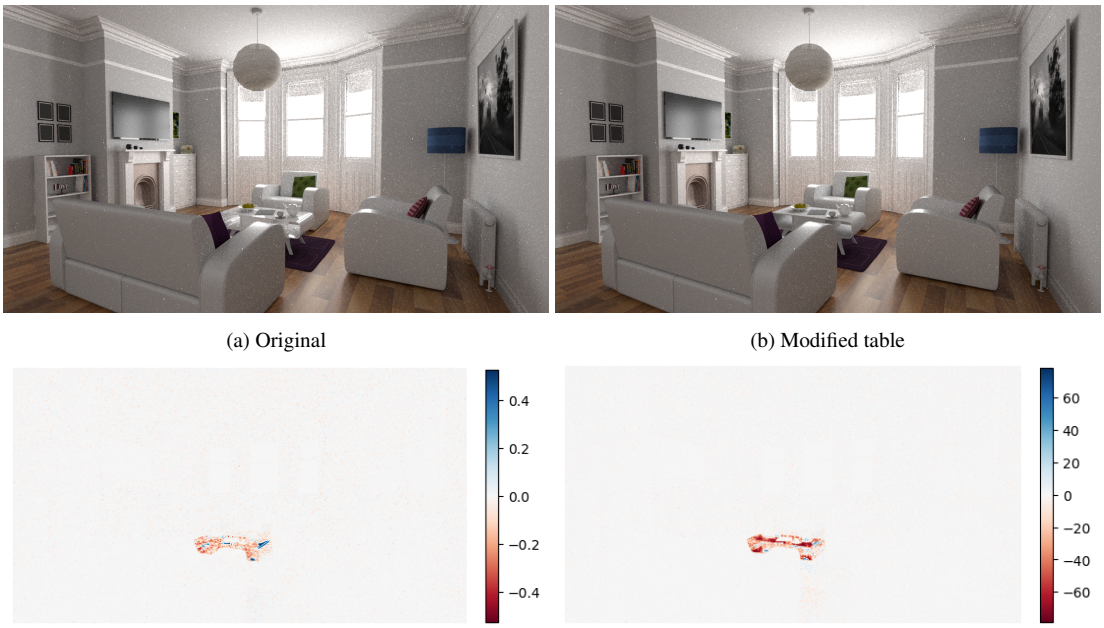
Comparing the change in cost for rendering the red box to the blue box scene, we can see how the costs of the paths between the wall and the box are affected. As the blue box reduces the throughput of paths bouncing from the red wall, it gets terminated earlier due to lack of throughput and Russian roulette.

In the following example, we look at the "White Room" scene provided by Benedikt Bitterli [Bit16]. In this scene, we replace the material of the center table (substrate material) with the same material as the table leg (matte material). The scenes are rendered with 256 spp. each and the metrics are evaluated with 32 spp. All the images are rendered with a maximum path depth of 33 in Figure 3. Due to the presence of few outliers, the absolute values in the difference costs are clipped after the 99.9th percentile.

As opposed to just changing the material's color, changing material properties additionally affects the path's trajectory. This change can lead to paths taking more or less expensive routes. At the same time, these changes can lead to paths terminating earlier. Looking at the cost difference, we can see how the change in the table's reflection affected the intersection cost. The places with high specular reflection had their costs increased when replaced with a matte material, whereas the regions without the specular highlights had their costs reduced. As the intersection cost is more representative of the actual computational cost, we can further see how significant the change is in estimated cost.

Spotting problematic areas

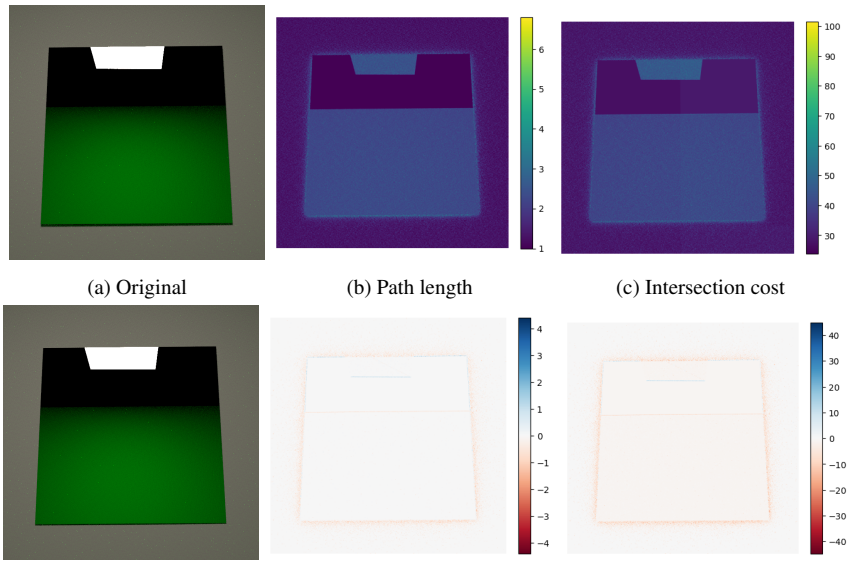
Looking at the metric visualization, we can see potential problem areas that are otherwise hidden while rendering the actual image. Take a simple example of a room with a mirror as shown in Figure 4. Here, looking at the actual render, we might not see anything unusual, but after visualizing our metrics, we can see an unusually high cost around the mirror's borders. It is caused due to the small gap between the wall and the mirror. This gap causes multiple rays to be trapped. To address this problem, we move the mirror closer to the wall. As a result, fewer paths get stuck between the wall and the mirror. Comparing the costs, we can see that we have reduced the estimated cost for rendering this scene.



(a) Original (b) Modified table

(c) Path length difference (d) Intersection cost difference

Figure 3: White room scene where the table is rendered with (a) substrate (mix of glossy and diffuse properties) material and (b) matte material matching the table legs. The cost difference in rendering the matte table scene is shown for (c) path length and (d) intersection cost. We can see how changing from a substrate to matte material increased the costs near the specular highlights in (a) the original image but deceased the costs in other regions.



(a) Original (b) Path length (c) Intersection cost

(d) Modified (e) Path length difference (f) Intersection cost difference

Figure 4: (a) Rendered image of mirror hanging on a wall and its corresponding (b) path length and (c) intersection costs (Equation 1). This scene is (d) modified to reduce the gap between the wall and the mirror. The difference in (e) path length and (f) intersection costs after performing this modification. The decrease in cost is due to less paths being trapped between the wall and the mirror.

6. Responsible Research

Due to the nature of path tracing, very few ethical considerations need to be taken. However, it does not imply that no ethical concern needs to be considered. As with any other research, high importance is given to ensuring that the data is not silently tampered with to match the expected results. Our work explicitly mentions where the data has been altered and why. In all cases, it is for visualization reasons, as a few outliers affect the range of colors utilized in color maps. Further, we mention the outlier removal technique applied for this purpose.

As our method is built on top of the widely known, open-source PBRT v3 renderer [PJH16], it is straightforward to re-implement our method. Furthermore, to address the reproducibility of our methods, we include the implementation details in Section 4. Additionally, all our test scenes except the scene with the mirror are extended from openly available scenes provided by PBRT [PJH16] and Benedikt Bitterli [Bit16]. Any alterations in the scenes are mentioned in Section 5.

7. Discussion

We have shown how our method can be used to estimate the cost of rendering a pixel and compare the effects of changing scene variables. It can be used to compare the cost contribution of changes in scene variables and highlight problematic areas that might need to be addressed. However, our method currently only supports visualizing the contribution of intersection tests to computational time. Although it can significantly contribute to computational time, it is not the only contributor. A few other contributors would involve loading memory-intensive textures, complex material properties and other rendering engine specific constraints. Developing a cost function to account for those factors can provide a more accurate representation of time spent per pixel.

Knowing the areas of computational hotspots defined by the metric might not always be helpful for an artist. Knowing the cause of the hotspots is beneficial as it will enable the artists to reduce the computational cost by addressing the cause. For example, path throughput can be defined as a metric to reason why the path length is high. However, it depends on how the rendering engine is built to determine the metrics. Suppose the rendering engine performs the Russian roulette calculation for path termination based on a fixed number. In that case, path throughput might not be a helpful metric to show how it affects path length.

In the depicted results, the sample rate per pixel is significantly lower than what would be used in practice for rendering a scene for an actual render of the scene. Still, it highlights the regions of high computational, albeit a noisy version of it. Due to the low sampling rate, it is significantly faster to render and potentially enables progressive rendering. It can also be extended to move the camera to make an interactive progressive cost map of the scene from a single movable viewpoint.

8. Conclusions and Future Work

We have developed a method for measuring and visualizing the costs for rendering a pixel for unidirectional path tracers and metrics for defining these costs. We showed that the intersection metric corresponds well to showing computational hotspots for the scenes where the intersections are still the highest contributor to computational time. Furthermore, the path length metric can be used along with the intersection metric to reason about high costs. These metrics can spot computationally expensive regions in a scene that might need to be addressed. To aid that, metrics can be used to compare similar scenes and get a better insight into the effects of changing scene variables on computational cost. This insight enables the user to perform clever modifications on the scene to reduce the computational cost.

In the future, this visual toolkit may be extended to support more metrics for a broader range of analyses and cost estimation. Additionally, the data collection method can be modified to collect the costs of the scene rather than for a single viewpoint.

9. Acknowledgements

We would like to thank Matt Phar, Wenzel Jakob and Greg Humphreys for their work on the PBR book and the PBRT v3 renderer [PJH16]. We would additionally like to thank Benedikt Bitterli [Bit16] for providing the sample scenes that could be run on PBRT v3.

References

[ABCC02] ARONOV B., BRÖNNIMANN H., CHANG A. Y., CHIANG Y.-J.: Cost prediction for ray shooting. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry* (New York, NY, USA, 2002), SCG '02, Association for Computing Machinery, p. 293–302. URL: https://doi.org/10.1145/513400.513438, doi:10.1145/513400.513438. 2

[AK90] ARVO J., KIRK D.: Particle transport and image synthesis. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (1990), pp. 63–66. 1, 2

[Bit16] BITTERLI B.: Rendering resources, 2016. https://benedikt-bitterli.me/resources/. 4, 6

[DNL*17] DENG Y., NI Y., LI Z., MU S., ZHANG W.: Toward real-time ray tracing: A survey on hardware acceleration and microarchitecture techniques. *ACM Comput. Surv. 50*, 4 (aug 2017). URL: https://doi.org/10.1145/3104067, doi:10.1145/3104067. 1

[GFE*12] GRIBBLE C., FISHER J., EBY D., QUIGLEY E., LUDWIG G.: Ray tracing visualization toolkit. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2012), I3D '12, Association for Computing Machinery, p. 71–78. URL: https://doi.org/10.1145/2159616.2159628, doi:10.1145/2159616.2159628. 2

[HM08] HUNT W., MARK W. R.: Ray-specialized acceleration structures for ray tracing. In *2008 IEEE Symposium on Interactive Ray Tracing* (2008), pp. 3–10. doi:10.1109/RT.2008.4634613. 2

[Jen01] JENSEN H. W.: *Realistic image synthesis using photon mapping*. AK Peters/crc Press, 2001. 2

[Kaj86] KAJIYA J. T.: The rendering equation. In *Computer Graphics* (1986), pp. 143–150. 1

[Lum17] LUMBERYARD A.: Amazon lumberyard bistro, open research content archive (orca), July 2017. http://developer.nvidia.com/orca/amazon-lumberyard-bistro. URL: http://developer.nvidia.com/orca/amazon-lumberyard-bistro. 1

[NHB17] NICHOLAS HULL K. A., BENTY N.: Nvidia emerald square, open research content archive (orca), July 2017. http://developer.nvidia.com/orca/nvidia-emerald-square. URL: http://developer.nvidia.com/orca/nvidia-emerald-square. 1

[PJH16] PHARR M., JAKOB W., HUMPHREYS G.: *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016. 1, 2, 3, 6

[RKJ96] REINHARD E., KOK A. J. F., JANSEN P. W.: Cost prediction in ray tracing. In *Rendering Techniques '96* (Vienna, 1996), Pueyo X., Schröder P., (Eds.), Springer Vienna, pp. 41–50. 2

[SHP*18] SIMONS G., HERHOLZ S., PETITJEAN V., RAPP T., AMENT M., LENSCH H., DACHSBACHER C., EISEMANN M., EISEMANN E.: Applying visual analytics to physically based rendering. *Computer Graphics Forum* (2018). URL: http://graphics.tudelft.nl/Publications-new/2018/SHPRALDEE18. 2

[VSM*18] VASIOU E., SHKURKO K., MALLETT I., BRUNVAND E., YUKSEL C.: A detailed study of ray tracing performance: render time and energy cost. *The Visual Computer 34* (06 2018). doi:10.1007/s00371-018-1532-8. 2

[WMG*09] WALD I., MARK W. R., GÜNTHER J., BOULOS S., IZE T., HUNT W., PARKER S. G., SHIRLEY P.: State of the art in ray tracing animated scenes. *Computer Graphics Forum 28*, 6 (2009), 1691–1722. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01313.x, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2008.01313.x, doi:https://doi.org/10.1111/j.1467-8659.2008.01313.x. 1

[WS95] WALTER B., SHIRLEY P.: Cost analysis of a ray tracing algorithm, 07 1995. doi:10.13140/RG.2.2.28794.03526. 2

[WW03] WIMMER M., WONKA P.: Rendering time estimation for real-time rendering. In *Proceedings of the 14th Eurographics Workshop on Rendering* (Goslar, DEU, 2003), EGRW '03, Eurographics Association, p. 118–129. 2