Photon: A Modular, Research-Oriented Rendering System

Tzu-Chieh Chang
National Taiwan University

Ming Ouhyoung
National Taiwan University

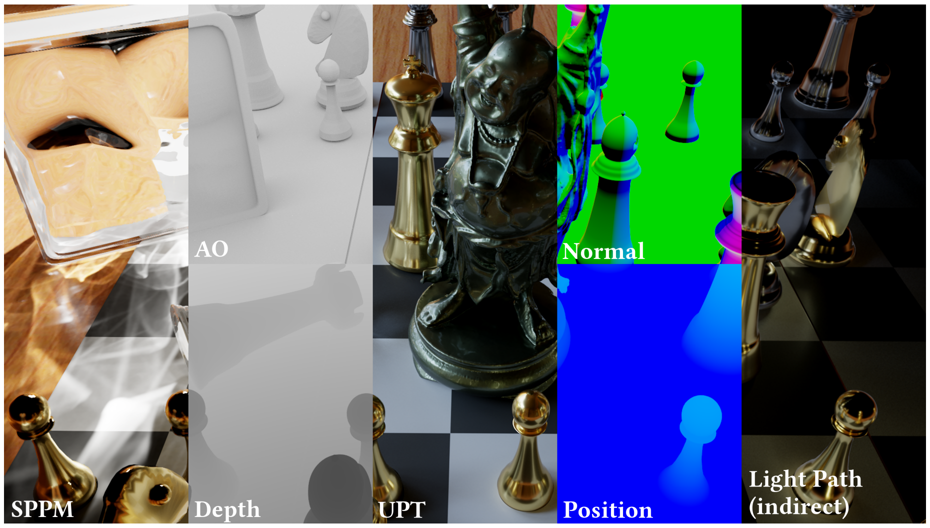


Figure 1: Several visualization techniques implemented using the building blocks provided by our system. In the center, the buddha features Belcour's BSDF model [Belcour 2018], while the left one, the glass brick induces caustics that can be rendered efficiently using particle tracing methods.

CCS CONCEPTS

• **Computing methodologies → Rendering**.

KEYWORDS

rendering, system, framework, modular, research, physically based

ACM Reference Format:
Tzu-Chieh Chang and Ming Ouhyoung. 2019. Photon: A Modular, Research-Oriented Rendering System. In *Proceedings of SIGGRAPH '19 Posters.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3306214.3338586

1 INTRODUCTION

To develop a graphics project with ease and confidence, the reliability and extensibility of the underlying framework are essential. While there are existing options, e.g., pbrt-v3 [Pharr et al. 2016] and Mitsuba [Jakob 2010], they either focus on education or not being updated for a long time. We would like to present an alternative solution named *Photon*.

Photon [Chang 2019] is an open-source and cross-platform[1] rendering system that, in its core, is designed with modularity and research in mind. The goal of our system is to provide a set of building blocks to facilitate the implementation of new rendering algorithms, as well as a unified foundation for comparing the performance of different methods. In addition, our system comes with both CLI and GUI applications, and the later is particularly useful for interactive visualization of the rendering progress. A Blender add-on is also available for content creation and can export 3-D assets to our scene description format.

In this article, we first propose a tri-layer software architecture that encapsulates common graphics concepts as independent modules, while maintaining the ability to fine-tune the behavior of each submodule. Secondly, we demonstrate the flexibility of the architecture by implementing several rendering algorithms, then showcasing some useful features. Finally, we conclude with a general comparison between similar systems and planned features for future releases.

2 THE TRI-LAYER ARCHITECTURE

We consider a rendering system as an assembly line that gradually transforms scene description data into 2-D imageries. As a result, each layer in our architecture corresponds to data in a specific state (Figure 2). The top layer is a COM-like[2] interface for storing scene description and procedural commands. Below it is a layer that translates raw data (scene description) into cooked data, where we borrow the concept of *content cooking* from Unreal Engine. To the best of our knowledge, Photon is the first among similar systems that formally introduce the cooking process as a separate architectural layer. It turns out that the introduction of such layer helps to

[1]Written primarily in C++. Currently supports Windows, Linux, and macOS.

[2]Component Object Model (COM) introduced by Microsoft.

SIGGRAPH '19 Posters, July 28 - August 01, 2019, Los Angeles, CA, USA
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6314-3/19/07.
https://doi.org/10.1145/3306214.3338586

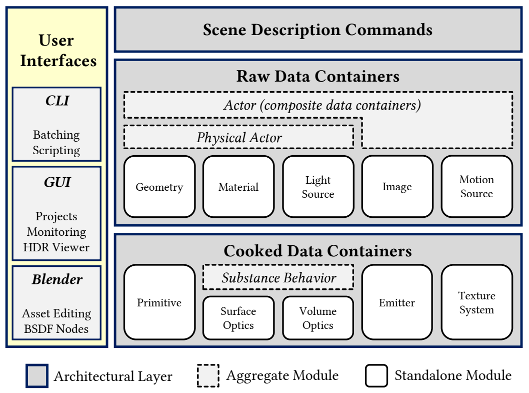


Figure 2: A high-level overview of the tri-layer architecture. Note that user interfaces typically act as content providers to the top layer, or data receivers from the bottom layer.

simplify logics, and opens an opportunity for project developers to natively alter scene data without intervening lower level modules.

3 SELECTED IMPLEMENTATIONS

Three flavors of photon mapping (PM [Jensen 1996], PPM [Hachisuka et al. 2008], SPPM [Hachisuka and Jensen 2009]) and an adaptive image sampler [Dammertz et al. 2010] are implemented; a shader node system that builds upon the scene description layer is also demonstrated.

3.1 Rendering Algorithms

The building blocks for photon mapping methods are a light path handler, an eye path handler, and an acceleration structure for photons. They are all highly templatized (in the style of C++ STL). Notably, the eye path handler consumes a policy object that controls the behavior of the underlying path sampling scheme, e.g., whether to branch path on intersections and the BSDF type to trace. This makes it possible to implement all three methods in a concise way (Figure 3b). Similarly, the adaptive image sampler makes use of a work scheduler along with the stepper camera measurement estimator. The *stepper* estimator controls sampling frequency on the image plane which is essential to the algorithm (Figure 3a, top).

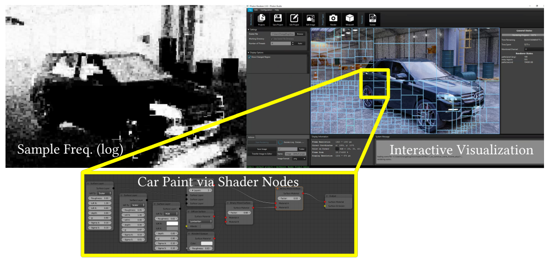
3.2 Shader Nodes

The system can automatically generate language bindings[3] for the scene description layer. Our Blender add-on includes a shader node system that is implemented by utilizing the Python binding (Figure 3a, bottom).

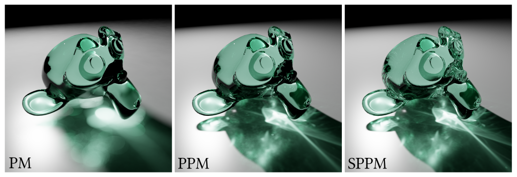
4 COMPARISONS AND FUTURE WORK

We compare our system with pbrt-v3, Mitsuba, Tungsten [Bitterli 2016], and SORT [Jiayin 2016] in three different aspects (Figure 4). In the comparison, although our system is weaker in document and features, we have a relatively young codebase and is quickly catching up its predecessors. Furthermore, our large number of unit tests allow developers to constantly check for consistent system behavior.

[3]Currently supports Python and Java.



(a) Adaptive sampling and shader nodes.



(b) Particle tracing methods. Note the differences in caustics and refraction.

Figure 3

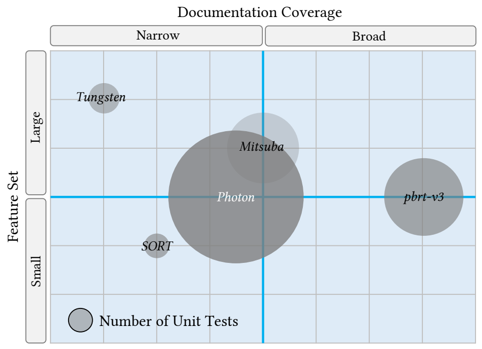


Figure 4: General comparison between similar systems. Darker shade indicates higher development activity. (We are aware of other open-source systems such as *LuxRender* and *appleseed*; not included as they are production oriented.)

In the future, we plan to incorporate more rendering algorithms, support other scene description formats for inter-system cooperations, and become a better open-source software.

REFERENCES

Laurent Belcour. 2018. Efficient Rendering of Layered Materials using an Atomic Decomposition with Statistical Operators. *ACM Transactions on Graphics* 37, 4 (2018), 1. https://doi.org/10.1145/3197517.3201289
Benedikt Bitterli. 2016. Tungsten renderer. https://benedikt-bitterli.me/tungsten.html.
Tzu-Chieh Chang. 2016–2019. Photon renderer. https://github.com/TzuChieh/Photon-v2.
Holger Dammertz, Johannes Hanika, Alexander Keller, and Hendrik P. A. Lensch. 2010. A Hierarchical Automatic Stopping Condition for Monte Carlo Global Illumination.
Toshiya Hachisuka and Henrik Wann Jensen. 2009. Stochastic Progressive Photon Mapping. *ACM Trans. Graph.* 28, 5, Article 141 (Dec. 2009), 8 pages. https://doi.org/10.1145/1618452.1618487
Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. 2008. Progressive Photon Mapping. *ACM Trans. Graph.* 27, 5, Article 130 (Dec. 2008), 8 pages. https://doi.org/10.1145/1409060.1409083
Wenzel Jakob. 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.
Henrik Wann Jensen. 1996. Global Illumination Using Photon Maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*. Springer-Verlag, London, UK, UK, 21–30. http://dl.acm.org/citation.cfm?id=275458.275461
Cao Jiayin. 2016. SORT. https://jerrycao1985.github.io/SORT/.
Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering: From Theory to Implementation* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.