Scene Conversion for Physically-based Renderers

SIBGRAPI paper ID: 51

Fig. 1: Example of automatic scene conversion obtained with our system. *Coffee Maker* rendered with PBRT v3 (left). Rendering produced by Mitsuba (center) and LuxRender (right), using converted scenes (from PBRT v3) for these rendering systems.

***Abstract*—Physically-based rendering systems use proprietary scene description formats. Thus, by selecting a given renderer for the development of a new technique, one is often constrained to test and demonstrate it on the limited set of test scenes available for that particular renderer. This makes it difficult to compare techniques implemented on different rendering systems. We present a system for automatic conversion among scene description formats used by physically-based rendering systems. It enables algorithms implemented on different renderers to be tested on the same scene, providing better means of assessing their strengths and limitations. Our system can be integrated with existing development and benchmarking APIs, lending to full orthogonality among algorithms, rendering systems, and scene files.**

I. Introduction

Monte Carlo ray tracing is currently the only practical solution for simulating global illumination effects in complex environments. Due to its high computational cost, several techniques have been introduced to reduce rendering time through improved sampling [1], [2] and reconstruction strategies [3]–[6]. When developing such new techniques, researchers often implement them on top of existing rendering systems as a way of leveraging available infrastructure to perform functions (*e.g.*, ray-traversal acceleration, ray-primitive intersections, etc.) that are orthogonal to the proposed methods.

Unfortunately, the various rendering systems use proprietary scene description formats. While modeling visually-pleasing scenes requires significant artistic skills, manual conversion between proprietary formats requires knowledge of the specific formats and tend to be extremely time consuming (up to several days per scene [7]). Thus, by selecting a given rendering system one is often constrained to test and demonstrate the proposed techniques on the limited set of test scenes available for that renderer. This apparently simple limitation has profound implications, as it constrains a direct comparison between Monte Carlo (MC) rendering techniques that have been implemented using different rendering systems. In this case, one often has to compare the quality of algorithms using disjoint sets of scenes, which, is not the ideal case.

We present *a system for automatic conversion among scene file formats used by Monte Carlo physically-based rendering systems*. Our solution significantly expands the repertoire of scenes available for testing, validation, and benchmarking of MC rendering algorithms. Currently, our system handles conversions among PBRT v3 [8], Mitsuba [9], and LuxRender [10], which are three of the most popular physically-based renderers (PBR). Extending it to support additional renderes is straightforward. Our solution (discussed in Section III) consists of *importing* any source scene description into a canonical representation, which can then be *exported* to other scene formats. Figure 1 illustrates the use of our system to perform automatic conversion of a scene represented in the PBRT v3 format. The images at the center and on the right show, respectively, the renderings produced by Mitsuba and by LuxRender, from converted scenes files. Note the correct representation of the various materials (glass, plastic, and metal).

Our work does not introduce a new physically-based rendering technique per se. Instead, it falls in the area of *meta-research* systems, which are systems designed to facilitate and improve the research process. Meta-research systems are quite common in computer graphics [11]–[14] and computer vision [15]–[18], where they have led to significant progress in these fields. Recently, Santos et al. [11] introduced a framework for developing and benchmarking MC sampling and denoising algorithms. This is achieved by providing an API the decouples the developed techniques from the used rendering system. While it allows a technique to be tested on any rendering system that supports the proposed API, each rendering system is still constrained to a limited set of test scenes. Our system is orthogonal to and complements the

API described in [11], lending to full orthogonality, among algorithms, rendering systems, and scene files.

The **contributions** of our work include:

- A system for automatic conversion among scene file formats used by Monte Carlo physically-based rendering systems (Section III). It enables algorithms implemented on different rendering systems to be tested on similar scene descriptions, giving developers and end users a better assessment of the strengths and limitations of MC rendering techniques;
- A mechanism for achieving full orthogonality among MC rendering algorithms, rendering systems, and scene files (Section III). This is achieved in combination with the API provided in [11].

II. Related Work

To the best of our knowledge, no previous system has performed automatic scene conversion among the major MC rendering systems. Bitterli has converted 32 scenes of various complexities and origins from Tungsten to PBRT v3 and Mitsuba using some scripts [7]. These scripts, however, are specific for conversions from Tungsten to these two renderers and are not publicly available. Our system can convert scene descriptions among multiple systems and is freely available for download [19].

RenderToolbox3 is a MATLAB tool developed for assisting vision research [20]. It imports a scene containing geometric objects described as COLLADA XML files, and allows one to associate to them reflectance measurements from a MATLAB Psycophysics Toolbox [21]. Such reflectance measurements are converted to multispectral reflectance representations compatible to PBRT and Mitsuba. A script then renders the objects with the associated multispectral representations using PBRT or Mitsuba. RenderToolbox3 is a visualization tool for exploring the impact of different reflectance and illuminating properties on human perception. The system does not convert scene files among rendering systems.

A. Meta-Research Systems in Computer Graphics

Several systems have been developed to support research in computer graphics. Some well-known examples include Cg [14], Brook [13], and Halide [12]. Cg is a general-purpose programming language designed to support the development of efficient GPU applications, and has stimulated a lot of research efforts in shader-based rendering techniques [22]–[25]. Brook [13] is system for general-purpose computation that allows one to exploit the inherent parallelism of modern GPUs without having to deal with GPU architecture details. These kinds of systems were an inspiration that led to the development of CUDA [26]. Halide [12] is a system designed to optimize image-processing applications on multiple hardware platforms by separating the algorithm description from its schedule. The system has been recently extended to support differentiable programming for image processing and deep learning [27]. All these systems focus on generating efficient code while freeing the user from GPU architectural details. All goal, in turn, is to make high-quality scenes availability independent of one's choice of rendering system.

Santos et al. have recently presented a framework for developing and benchmarking MC sampling and denoising algorithms [11]. They use an API to decouple algorithms from rendering systems, allowing for the same algorithm to be tested on multiple rendering systems. By doing so, they also increase the set of scenes an algorithm can be tested with. However, in order to use a given test scene, the rendering system for which the scene was created would have to be used as well. Our scene-conversion solution complements the API described in [11], lending to a desirable full orthogonality among MC algorithms, rendering systems, and scene files.

III. Automatic Scene Conversion

Our system consists of two main components: an *import module* that reads an arbitrary scene file format and generates an equivalent description in a *canonical* scene representation; and an *export module* that takes our canonical representation and exports it to a target rendering system file format. The complete process is illustrated in Figure 2. Currently, our system supports *PBRT v-3* [8], *Mitsuba* [9], and *LuxRender* [10], as these are three of the most popular rendering systems. This architecture, however, is quite flexible. Supporting additional rendering systems only requires specializing the import and export methods to handle the new formats. Next, we describe the main components of our system.

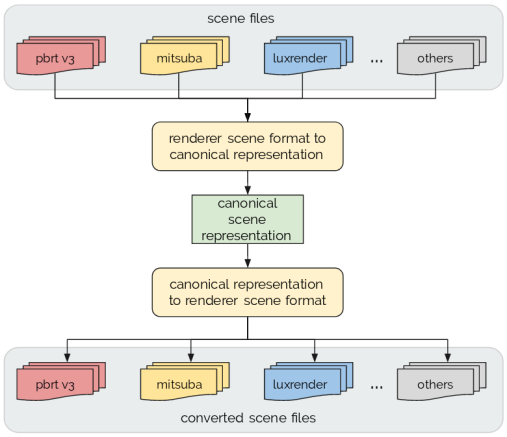


Fig. 2: Our scene conversion pipeline. An input scene description is imported into a canonical representation, which, in turn, can be exported to a target rendering system format.

A. The Import Module

Most physically-based renderers subdivide the scene description in two main sections: *scene-wide rendering options* and *world block*. The former defines the rendering settings, while the latter describes the scene geometry and materials. The import module parses the input scene files and translates each directive into a canonical representation. Since rendering systems use proprietary file format, both the import and export modules have to be specialized for each renderer.

PBRT and LuxRender scene descriptions consist of structured text statements. We generated parsers for these systems using PLY [28], a Python implementation of Lex and Yacc. Mitsuba, in turn, is a heavily optimized, plugin-oriented renderer. Its file format is, essentialy, an XML description of which plugins should be instantiated with the specified parameters. Since there are several XML-parsing libraries for Python, we chose to use ElementTree [29], a Python XML parsing tool.

B. Canonical Scene Representation

While most renderers have a similar structure, they differ in a few supported features and in the parameters used to configure the rendering process. Thus, we need a canonical representation that covers the features supported by all renderers. COLLADA [30] is an XML schema intended as a representation for exchanging digital content among graphics applications. However, COLLADA files only include information about scene geometry. No information about other rendering options, such as camera positioning or integration techniques, is available. In order to establish a common ground for conversion, we defined a canonical scene representation. It is illustrated in Figure 3 and can easily extended to incorporate any directives not covered in our current implementation.

Our canonical representation mirrors the general structure of scene files and divides the scene data into scene-wide *rendering options* and *world block*. This is illustrated in Figure 3, where the attributes stored for each scene component are shown on the rectangles on the right.

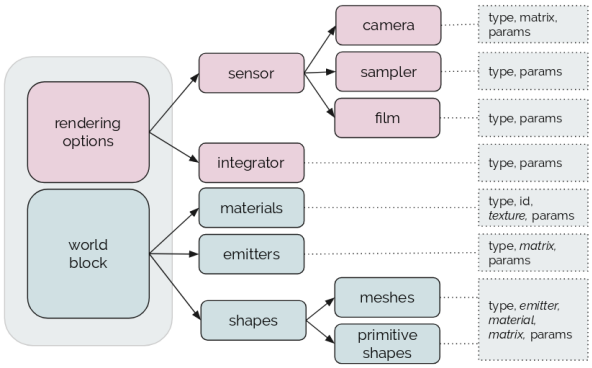


Fig. 3: Structure of our canonical scene representation, consisting of rendering options and scene data. The attributes stored for each component are shown on the rectangles on the right.

The *Rendering Options* specify the integration and sampling techniques used for rendering, as well as camera and film properties. These include, for instance, camera position, camera matrix, image resolution, field of view, etc. Table I summarizes all types, parameters, and additional attributes associated with each component of our canonical scene representation.

The *World Block* describes the materials, global emitters, and shapes present in the scene. A *material* (*e.g.*, glass, plastic, metal, etc.) may have one or more associated textures. *Global emitters* represent all kinds of light sources, except area light sources, which are represented as shapes. These include conventional environment, spot, directional, and point light sources, as well more specific ones such as *sun* and *sky*. A *shape* can be a polygonal mesh or a geometric primitive such as a rectangle, disk, cube, or sphere, for instance.

Component	Type	Parameters	Others
camera	environment, orthographic, perspective, realistic	focal distance, fov, lens aperture, near/far clip, shutter open/close	view matrix
sampler	halton, random, sobol, stratified	samples per pixel, scramble	-
film	hdr, ldr	file extension (png, ...), filter, image height, image width	-
integrator	bidirectional path tracer, direct lighting, metropolis light transport, path tracer, photon mapping	max depth, number of iterations, number of Markov chains, photon count, photon mapping lookup radius, russian roulette depth	-
materials	glass, matte/diffuse, metal, substrate/glossy, translucent, uber	η, id, IOR, k, kd, ks, reflectance, roughness, transmittance, ...	texture (id, type, params)
emitters	directional, distant, environment mapping, sky, spot, sun	filename, from (origin), intensity, radiance, to (direction)	model matrix
shapes	mesh (ply/obj)	filename	model matrix, area emitter, unnamed material
	rectangle, disk, triangle mesh, cube, sphere	center, normals, points, radius, uv mapping, ...	

TABLE I: Types, parameters and additional attributes of the components in our canonical scene representation (Figure 3).

C. The Export Module

The export module is at the core of our system. While the import module deals with a single proprietary scene representation at a time, the export module has to map between materials and scene properties from two proprietary representations. In this case, there are several delicate cases to consider. Matrix transformations, native shapes, environment mapping coordinates and, mostly, materials are some of the components that vary greatly between renderers. In several situations, there is no direct mapping between them. Still, our system should provide an output representation that, once rendered with the target system, best approximates the results obtained by the source rendering system with the input scene description. Achieving such results required extensive experimentation with parameters of the various systems. Next, we discuss a few relevant aspects one should consider.

Matrix Conversion: There are several issues to consider when converting matrices between renderers. Do the two

renderers use different coordinate systems (either left-handed or right-handed)? Do they represent matrices in the scene file using a direct representation or its inverse-transpose? How is the object-world transformation represented for shapes?

Mitsuba uses a right-hand coordinate system, while PBRT and LuxRender use a left-hand one. This means that, when converting between Mitsuba and the other two, one has to mirror the x-axis of all camera matrix transformations. This is also the case for environment map positioning and object-world transformations. Moreover, Mitsuba's scene files contain a world-to-camera transformation matrix (*i.e.*, view matrix), while PBRT and LuxRender scene files use the view matrix inverse transpose.

Material Conversion: materials are the most delicate aspect of scene conversion. Materials have spectral and roughness properties that absolutely must be correctly mapped. However, most renderers have very different implementations for common subsurface scattering models (BSDFs), making it hard to predict the mapping between the parameters of two such implementations.

Mitsuba uses a more physics-oriented approach: a material can be diffuse, conductor, dielectric, plastic, translucent, or a bumpmap. It also has other types of materials, but those are not supported in the current implementation of our system. The material type in Mitsuba changes as the material contains any form of surface roughness, becoming a "rough" version of itself (for instance, a rough metal becomes a roughconductor). PBRT and LuxRender materials have roughness parameters, making it unnecessary to change the material's name.

To represent the material's reflectance, PBRT and Mitsuba use one index of refraction (η) and one absorption coefficient (k) per color channel. LuxRender, however, uses a so-called "*Fresnel texture*", specifying a single value of η and k for all channels. Alternatively, LuxRender allows the specification of a single RGB color value for the material's reflectance. Therefore, correctly converting metal colors between LuxRender and PBRT or Mitsuba is not well defined, and is not supported in the current implementation of our system.

Shape Conversion: shape directives can be split into two categories: *primitive shapes*, which can be used to specify primitives such as *rectangles, disks, cubes*, and *spheres*; and *3D meshes*, which are stored in external files. Converting primitive shapes requires more attention than converting external 3D meshes. Mitsuba has directives for rectangle, disk, cube and sphere, while PBRT and LuxRender do not. Mitsuba's primitives are defined by some parameters (*e.g.*, vertex positions, radius) which can be modified by a transformation (model) matrix. To reproduce these primitives in PBRT and LuxRender, an *internal triangle mesh* must be used. This is done by specifying the position, normal, and texture coordinates for each vertex in the mesh representing a given primitive. One should note that these internal meshes do not use the same representation as the 3D meshes stored in files.

Converting PBRT and LuxRender internal triangle meshes into Mitsuba primitive shapes is a more involving process. Since Mitsuba's primitives have predefined coordinates, converting vertices from PBRT or LuxRender internal meshes into these coordinates requires obtaining the transformation matrix that maps PBRT or LuxRender vertices to Mitsuba's predefined points. Our system takes care of this automatically.

Converting external 3D meshes is simple, as all rendering systems have directives for this purpose. PBRT, however, does not support Object File Wavefront 3D (.obj) files. In this case, our system issues a warning, making the user aware of the need to convert .obj files off-line.

Global Emitter Conversion: global emitters can be used to emulate environment lighting, such as the sun, the sky, or an environment map. Converting global emitters can be tricky, mainly because different rendering systems do not implement the same algorithms and directives. For instance, Mitsuba and LuxRender implement *sun* and *sky* directives, while PBRT does not. A sun directive can be simulated in PBRT using a distant light. A sky directive can be simulated using an environment map of a clear sky. While PBRT and LuxRender access environment maps using spherical coordinates, Mitsuba uses a latitude-longitude format. Thus, a conversion between the two representations is required.

Converting a PBRT distant light into a sun directive for Mitsuba or LuxRender is straightforward. However, converting a PBRT environment map into a sky directive lends to an ambiguous situation, as the converter would require additional information to decide whether the environment map should be treated as a regular environment map, or as a sky directive. Our system solves this ambiguity by asking the user if the environment map should be converted to a sky emitter.

IV. Results

Our system is available on-line [19]. We have tested it on a large number of scenes, including the 32 scenes available at Bitterli's rendering resources website [31]. Here, we include a few examples to illustrate its results on scenes that explore different types of materials, 3D meshes and primitive shapes, image and procedural textures, and various lighting styles. They include most elements typically found in scenes used by physically-based rendering systems. The time required to convert a scene is about 0.5 seconds on a typical PC (Intel i5 3.8 GHz). The scenes were rendered using Mitsuba 0.5.0, PBRT v3, and LuxRender v1.6 on Ubuntu 14.04 LTS. All scenes were rendered using between 5,000 and 8,000 samples per pixel (spp). For any given scene, the same number of samples per pixel was used with all rendering systems.

Figure 1 shows a coffee maker containing various materials, including glass, plastic, and metal, as well as textures. The input scene description was provided in the format for PBRT v3, whose rendering is shown on the left. The images at the center and on the right were produced by Mitusuba and LuxRender, respectively, from scene representations automatically converted by our system. Note how the object details have been faithfully preserved in these renderings.

The *Wooden Staircase* scene (Figure 4) contains many geometric objects and textures. A LuxRender scene description was provided as input and its rendering is shown in (a). The

(a) LuxRender (b) PBRT v3 (c) Mitsuba

Fig. 4: *The Wooden Staircase* scene. Input scene description for LuxRender (a). Renderings produced by PBRT v3 (b) and Mitsuba (c), from scene descriptions converted by our system.

images shown in (b) and (c) were produced by PBRT v3 and Mitsuba, respectively, from scene representations automatically converted by our system.

The *Teapot* scene (Figure 5) contains a shiny object, environment lighting, and a procedural texture. The input scene description was also provided in the LuxRender format. Figures 5b and 5c show the renderings produced by PBRT v3 and Mitsuba, respectively, from scene descriptions converted by our system.

Figure 6 shows two scenes, *Veach Bidir Room* and *Cornell Box*. The first includes caustics, while the second only contains diffuse surfaces. A Mitsuba scene description was provided as input for each of these scenes, whose renderings are shown on the first column of Figure 6. Columns (b) and (c) show, respectively, the renderings produced by PBRT v3 and LuxRender using scene descriptions converted by our system.

A. Discussion

The renderings produced by different rendering systems may exhibit significant differences in color or shading due to features unsupported by some renderers. For instance, consider the use of a light source to emulate the sun. In PBRT and LuxRender, this directive is implemented as a distant white light. Mitsuba, in turn, emulates the sun using a distant environment light implemented according to a technique described in [32], which produces a warm-colored, distant light source. Thus, when the sun directive is used, Mitsuba renderings present a different color compared to the other two. This situation is illustrated in Figure 7.

LuxRender does not properly handles a combination of sun directive and local light sources. This is illustrated in Figure 8c, where hard shadows have turned soft. The difference in colors are due to the sun directive, as discussed above.

(a) Mitsuba (b) PBRT v3 (c) LuxRender

Fig. 8: *Little Lamp* scene. Input scene description for Mitsuba (a). Renderings produced by PBRT v3 (b) and LuxRender (c), from scene descriptions converted by our system.

The supplemental materials included with this submission show all the examples presented in the paper plus additional ones. We would like to encourage the reader to explore them, where one can inspect the images at their original resolutions.

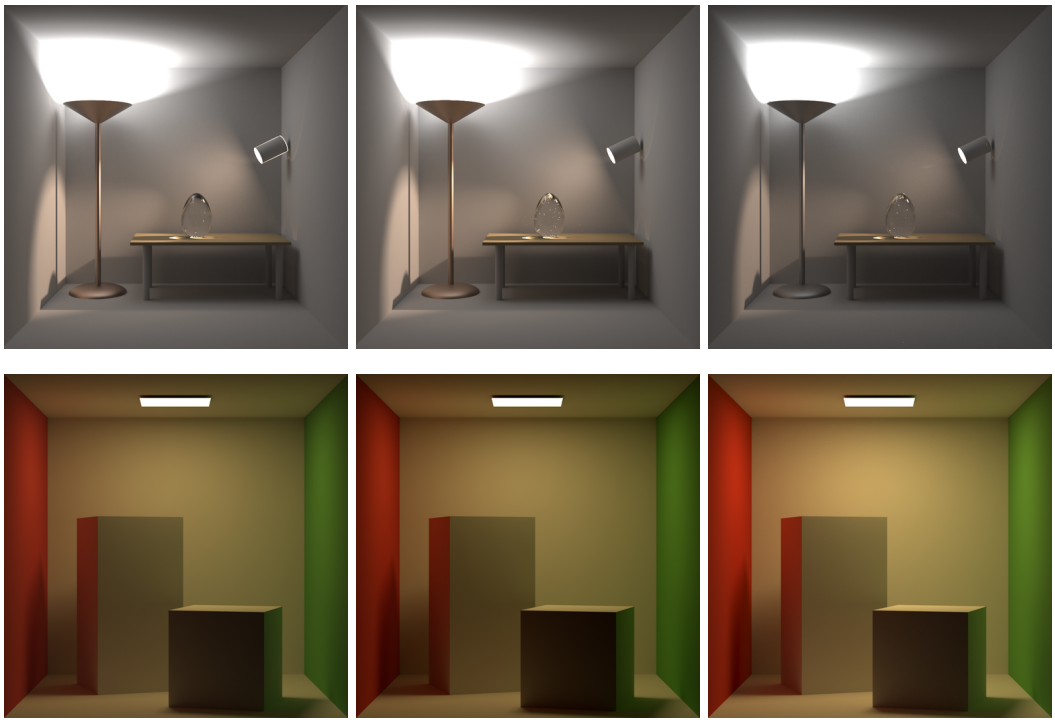
B. Limitations

Scene-description directives found in one rendering system but without correspondence in the other two renderers are not handled by our system. That is the case, for instance, of Mitsuba-only materials like *phong* and *blendbsdf*.

The current version of our system does not support the conversion of hair or participating media. As discussed in Section III, LuxRender treats material reflectance differently from PBRT and Mitsuba. Thus, properly converting metal colors to LuxRender is a challenging task, not currently supported by our system. This is illustrated in Figure 9, where the rendering of metal obtained from a scene converted to and rendered with LuxRender looks darker.

(a) LuxRender (b) PBRT v3 (c) Mitsuba

Fig. 5: *Teapot* scene. Input scene description for LuxRender (a). Renderings produced by PBRT v3 (b) and Mitsuba (c), from scene descriptions converted by our system.

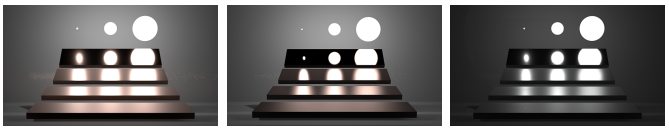


(a) Mitsuba (b) PBRT v3 (c) LuxRender

Fig. 6: *Veach, Bidir Room* (top) and *Cornell Box* (bottom). Input scene descriptions for Mitsuba (a). Renderings produced by PBRT v3 (b) and LuxRender (c), from scene descriptions converted by our system.

(a) LuxRender (b) PBRT v3 (c) Mitsuba

Fig. 7: *The Breakfast Room* scene. Input scene description for LuxRender (a). Renderings produced by PBRT v3 (b) and Mitsuba (c), from scene descriptions converted by our system. Mitsuba's sun directive produces a warm-colored lighting.



(a) PBRT v3 (b) Mitsuba (c) LuxRender

Fig. 9: *Veach, MIS*. Renderings by PBRT v3 (a), Mitsuba (b), and LuxRender (c). Converting metal colors to LuxRender is a challenging task, not currently supported by our system.

V. Conclusion

We presented a system for automatic conversion among scene file formats used by Monte Carlo physically-based rendering systems. It enables algorithms implemented using different renderers to be tested on similar scene descriptions, providing better means of assessing the strengths and limitations of MC rendering techniques. Our system can be easily integrated with the API recently introduced by [11], allowing researchers and developers to exploit full orthogonality among MC algorithms, rendering systems, and scene files.

We have demonstrated the effectiveness of our system by converting scene description among three of the most popular MC rendering systems: PBRT v3, Mitsuba, and LuxRender. Providing support to additional renderers only requires specializing the import and export modules described in Section III for the given renderers. Our system is freely available and we encourage developers to provide support for other renderers.

In the future, we would like to add support for the conversion of hair and participating media, as well as for other rendering systems. By documenting limitations and incompatibilities found among different renderers, our work might stimulate efforts to reduce these differences.

References

[1] D. Heck, T. Schlömer, and O. Deussen, "Blue noise sampling with controlled aliasing," *ACM Trans. Graph.*, vol. 32, no. 3, pp. 25:1–25:12, 2013.

[2] A. Pilleboue, G. Singh, D. Coeurjolly, M. Kazhdan, and V. Ostromoukhov, "Variance analysis for monte carlo integration," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 124:1–124:14, 2015.

[3] P. Sen and S. Darabi, "On filtering the noise from the random parameters in Monte Carlo rendering," *ACM Trans. Graph.*, vol. 31, no. 3, pp. 1–15, 2012.

[4] F. Rousselle, M. Manzi, and M. Zwicker, "Robust denoising using feature and color information." *Comput. Graph. Forum*, vol. 32, no. 7, pp. 121–130, 2013.

[5] N. K. Kalantari, S. Bako, and P. Sen, "A machine learning approach for filtering Monte Carlo noise," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 122:1–122:12, jul 2015.

[6] B. Bitterli, F. Rousselle, B. Moon, J. A. Iglesias-Guitián, D. Adler, K. Mitchell, W. Jarosz, and J. Novák, "Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings," *Computer Graphics Forum*, vol. 35, no. 4, pp. 107–117, jul 2016.

[7] B. Bitterli, "The tungsten renderer," 2014, https://benedikt-bitterli.me/tungsten.html.

[8] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering, from Theory to Implementation*, 3rd ed. Morgan Kaufmann, 2016.

[9] W. Jakob, "Mitsuba: Physically based renderer," 2014, http://www.mitsuba-renderer.org/.

[10] J.-P. Grimaldi and T. Vergauwen, "Luxrender v1.6," 2008, http://www.luxrender.net/.

[11] J. D. B. Santos, P. Sen, and M. M. Oliveira, "A framework for developing and benchmarking sampling and denoising algorithms for monte carlo rendering," *The Visual Computer*, vol. 34, pp. 765–778, 2018.

[12] J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand, "Decoupling algorithms from schedules for easy optimization of image processing pipelines," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 32:1–32:12, Jul. 2012.

[13] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for gpus: Stream computing on graphics hardware," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 777–786, Aug. 2004.

[14] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard, "Cg: A system for programming graphics hardware in a c-like language," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 896–907, Jul. 2003.

[15] D. Scharstein, R. Szeliski, and H. Hirschmller, "Middlebury Stereo Vision Page," 2002. [Online]. Available: http://vision.middlebury.edu/stereo/

[16] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, and R. Szeliski, "Middlebury Flow Accuracy and Interpolation Evaluation," 2011. [Online]. Available: http://vision.middlebury.edu/flow/eval/

[17] C. Rhemann, C. Rother, J. Wang, M. Gelautz, P. Kohli, and P. Rott, "Alpha matting evaluation website," 2009. [Online]. Available: http://www.alphamatting.com/eval_25.php

[18] M. Erofeev, Y. Gitman, D. Vatolin, A. Fedorov, and J. Wang, "Videomatting," 2014. [Online]. Available: http://videomatting.com/

[19] "PBR scene converter," https://github.com/lahagemann/pbr_scene_converter.

[20] B. S. Heasly, N. P. Cottaris, D. P. Lichtman, B. Xiao, and D. H. Brainard, "Rendertoolbox3: Matlab tools that facilitate physically based stimulus rendering for vision research," *Journal of Vision*, vol. 14, no. 2, p. 6, 2014.

[21] D. H. Brainard, "The psychophysics toolbox," *Spatial Vision*, vol. 10, pp. 433–436, 1997.

[22] F. Policarpo, M. M. Oliveira, and J. a. L. D. Comba, "Real-time relief mapping on arbitrary polygonal surfaces," in *Proc. the ACM Symposium on Interactive 3D Graphics and Games*, 2005, pp. 155–162.

[23] F. Policarpo and M. M. Oliveira, "Relief mapping of non-height-field surface details," in *Proc. ACM Symposium on Interactive 3D Graphics and Games*, 2006, pp. 55–62.

[24] C. Wyman, "An approximate image-space approach for interactive refraction," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1050–1053, 2005.

[25] M. M. Oliveira and M. Brauwers, "Real-time refraction through deformable objects," in *Proc. ACM Symposium on Interactive 3D Graphics and Games*, 2007, pp. 89–96.

[26] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," in *ACM SIGGRAPH 2008 Classes*, ser. SIGGRAPH '08. New York, NY, USA: ACM, 2008, pp. 16:1–16:14.

[27] T.-M. Li, M. Gharbi, A. Adams, F. Durand, and J. Ragan-Kelley, "Differentiable programming for image processing and deep learning in halide," *ACM Transactions on Graphics*, vol. 37, no. 4, 2018.

[28] D. Beazley, "Ply (python lex-yacc)," 2001. [Online]. Available: http://www.dabeaz.com/ply/

[29] "The ElementTree XML API," https://docs.python.org/3/library/xml.etree.elementtree.html.

[30] R. Arnaud and M. C. Barnes, *Collada: Sailing the Gulf of 3D Digital Content Creation*. AK Peters Ltd, 2006.

[31] B. Bitterli, "Rendering resources," 2016, https://benedikt-bitterli.me/resources/.

[32] A. J. Preetham, P. Shirley, and B. Smits, "A practical analytic model for daylight," in *Proc. SIGGRAPH '99*, 1999, pp. 91–100.