UCSD CSE 272 Assignment 3:

Final Project

The final project will be a project of your choice. You can implement some non-trivial rendering algorithms, or render some interesting scenes in lajolla or your own renderer. It can also be a rendering-related research project with a new algorithm or system. Below we provide some options for you to think about, but feel free to come up with your own ones.

Grading. The project will be graded based on two aspects: technical sophistication and artistic sophistication. You can have a project that does not have much technical component, but it renders beautiful images. Alternatively, you can have a project that produces programmer arts, but is highly technical. To encourage people to pursue ambitious projects, the more ambitious the project is, the less requirement that is to the completness of the project. For example, if you aim to work on a research idea that is novel enough to be published at SIGGRAPH, we will not judge you using the standard of a technical paper. It is sufficient to show your work and the initial prototypes and experiments you develop, even if the initial result does not show practical benefit yet.

Teaming up. For the final project, you can either do it alone or form a group of two people. We do expect that you'll do more difficult projects with two people, but only slightly so. As every software developer know, more people doesn't immediately lead to faster progress, so be careful. What teamwork buys you is probably more capacity to brainstorm ideas and the friendship.

For the research type projects, to avoid conflicts, if multiple groups want to work on the same project, we will consider merging the groups.

Logistics. Before 2/26, please let us know what you plan to do for the final project by submitting a 1-2 pages brief report to Canvas. Schedule a chat with us if you have any question. We will have a checkpoint at 3/11: send us a brief progress report in a few pages on Canvas (ideally provide results and images) describing what you did and what you plan to do next.

1 Implementation ideas

Following are projects that are less novel, but could produce cool images. They are usually about the implementation of the papers we talked about in the class. I did not sort them in terms of difficulty.

Energy-preserving microfacet BSDFs. Implement Heitz et al.'s multiple-scattering microfacet BSDF [17] in lajolla. Alternatively, implement the position-free variant [42]. As a bonus, compare them to the energy compensation technique introduced by Kulla and Conty, and show the pros and cons.

Normal map and displacement map filtering. Implement LEAN [36] and use it for rendering high resolution normal map in lajolla. Compare it to a high sample count reference. When does it work well and when does it fail? As a bonus, implement LEADR [10] for filtering displacement map. First you will need to implement displacement mapping in lajolla. This can be done in a preprocessing pass to convert meshes with displacement map textures into tesselated mesh, or in an on-the-fly manner [40]. Alternatively, implement Yan et al.'s normal map filtering technique [47] in lajolla.

Layered BSDF. Implement the position-free layered BSDF simulator from Guo et al. [14] in lajolla. As a starting point, maybe start with a two layer BSDF with no volumetric scattering in between. Implementing the unidirectional version is good enough. As a bonus, read Gamboa et al.'s work [11] and implement their sampling strategy.

Hair BCSDF. Implement Marschner's hair scattering model [29] in lajolla. You'll have to implement a ray-curve intersection routine (it is acceptable to reuse the one from Embree) and a hair BCSDF. You may want to read Matt Pharr's note on implement hair rendering in pbrt.

Thin-film iridescence BSDF. Implement Belcour's thin-film iridescence BSDF [6] in lajolla. Feel free to look at Belcour's source code.

Faster null-scattering using space partitioning. Implement Yue et al.'s kd-tree data structure [48] or uniform grid [49] for better upper bound of the extinction coefficients. As a bonus, implement progressive null tracking [31].

Emissive volumes. Add emissive volumes to your Homework 2 code. You might want to read Pixar's volume sampling paper from Villemin and Hery [41].

Efficient transmittance estimation. Implement Kettunen et al.'s unbiased ray marching estimator [19]. You can extend your volumetric path tracer in homework 2 for this. Replace your ratio tracker with the new ray marching estimator. Compare to the ratio tracker and analyze their variance reduction properties.

SGGX. Implement the SGGX microflake phase function [16] in lajolla. Use it for rendering something like anisotropic volume appearance or level of details.

BSSRDF. Implement a BSSRDF in lajolla. You can use the one from Christensen and Burley [9]. For importance sampling the BSSRDF, check out King et al.'s talk. You are encouraged to read pbrt's code.

Single scattering. Implement Chen et al.'s 1D Min-Max mipmap shadow mapping technique for rendering single scattering volumes [8]. You don't have to implement it on GPUs or show real-time performance.

Differentiable rendering. Implement edge sampling or warped area sampling in Slang (if you want you can do it in lajolla, but it's likely to be a lot of work).

Stratification. Implement low-discrepancy sampling in lajolla. A simpler starting point is Halton sequence. Read the related chapters in pbrt for the reference. Smallppm from Toshiya Hachisuka also contains a low discrepenacy photon mapper using Halton sequence.

Bidirectional path tracing. Implement a bidirectional path tracer in lajolla. Read the related chapters in pbrt for the reference. Another good reference code is smallpssmlt from Toshiya Hachisuka. As a bonus, implement the efficiency-aware multiple importance sampling from Grittman et al. [13].

Progressive photon mapping. Implement progressive photon mapping [15] in lajolla. Read the related chapters in pbrt. Another good reference code is smallppm from Toshiya Hachisuka. You can also implement the probabilistic variant from Knaus et al. [22].

Gradient-domain path tracing. Implement gradient-domain path tracing [20] in lajolla. A good reference code is smallgdpt written by me.

Metropolis light transport. Implement Kelemen-style Metropolis light transport [18] in lajolla. Read the relevant chapters in pbrt. Another good reference code is smallpssmlt and smallmmlt from Toshiya Hachisuka.

Manifold exploration. Implement Zeltner's specular manifold sampling [51] in lajolla. Start with a basic LSDE path with reflective objects (instead of refraction).

Optimal multiple importance sampling. Implement *optimal* multiple importance sampling using control variates from Kondapaneni et al. [23] in lajolla. Ideally, apply their method for a unidirectional path tracer or even a bidirectional path tracer (instead of just applying it for direct lighting) – what kind of data structure do you need for maintaining the required statistics?

Lightcuts. Implement stochastic lightcuts [50] from Yuksel et al. in lajolla for rendering scenes with millions of lights. As a bonus, implement the data-driven version from Wang et al. [43].

ReSTIR. Implement ReSTIR [7] in lajolla for rendering scenes with many lights or even ReSTIR PT [28]. You don't need to implement the temporal reuse, but you should explore spatial reuse.

GPU rendering. Port most of lajolla to Vulkan/DirectX 12/CUDA/OpenCL/Metal/Slang.

Build a game in Unreal Engine 5! Discuss what rendering technology you used in the game and why you used them.

2 Research project ideas

These projects are likely publishable in a conference or a journal if done well (you will likely need to work on it longer even after the course ends for this to happen though). They are for students who are more motivated and want to get into rendering research. If you choose to work on these projects, it is possible that they will not be finished at the end of the quarter. This is totally fine and you will still get high/full points if you show your work. We are happy to work with you after the quarter to finish the project if you did well and are interested (and if we have the capacity).

For anything below, feel free to reach out to us for clarification about details of the project. I did not sort them in terms of difficulty.

Differentiable hair rendering. Existing inverse hair rendering methods (e.g., [37]) do not consider the light scattering in the hair fiber, and this can lead to less realistic results. In this project, we will develop a differentiable renderer [27] that is capable of computing the derivatives of a rendering of hair with respect to the shading and geometry parameters, and use it for inferring hair reflectance and geometry from images. You can either implement this in Mitsuba 3 or Slang to leverage their automatic differentiation feature and GPU backends. For the scope of the final project, it is sufficient to only consider the hair BSDF parameters and ignore the curve geometry. To handle the geometry derivatives, you can likely adapt Bangaru et al.'s differentiable signed distance field rendering algorithm [4] (since the computation of ray-curve intersection relies on the distance between the ray and the curve).

Differentiable curved surfaces rendering. Most of the discontinuity handling methods in differentiable rendering currently focuses on triangle meshes and signed distance fields. In Computer Aided Design, parametric surfaces such as NURBS or Beizer patches are often used (see Bartosz Ciechanowski's blog post for an excellent introduction). Can you design a method to handle the discontinuity when differentiating curved surface rendering? Note that existing work handled this by first converting the geometry to triangle meshes [44], and we are aiming for an exact method without the conversion. You may find the technique of "implicitization" useful – it converts a polynomial parametric surface into an implicit surface, you can then apply our differentiable SDF rendering technique [4]. Same as above, you can either implement this in Mitsuba 3 or Slang to leverage their automatic differentiation feature and GPU backends.

Theoretical analysis of warped-area sampling. While we have a sketch of correctness analysis in the warped-area sampling papers [5, 4, 45], the theoretical properties of the velocity field interpolation are not very well-understood. What is the optimal interpolation? When will the integral diverge? If you choose to work on this, you likely don't need to write much code (other than to verify your intuition) and it's mostly about writing down the formalism.

Guided tri-directional path tracing. From Veach's path-space formulation, it is clear that in addition to tracing rays starting from eyes or lights, it should also be possible to trace rays starting from an arbitrary point in the scene (we explored a similar method in 2017 [2]). However, such *tri*-directional path tracing method is difficult to apply in practice since it is difficult to decide where should we start tracing the ray. In this project, we will design a path guiding algorithm [24, 32] that builds a 5D data structure to decide which point and direction the path should start with. In the first phase, we will start tracing rays randomly, and record their contributions in the 5D data structure. We will then importance sample the recorded contribution and iteratively update the data structure. Similar methods have been applied to differentiable rendering (e.g., [46]), but it has not been applied in forward rendering algorithms yet. For the scope of the final project, it is not necessary to combine your method with the camera and light subpaths. Ultimately, it would be super cool to combine the tri-directional path tracer with VCM/UPS and use data to decide with subpath to use [13].

Motion-aware path guiding for animation rendering. Path guiding algorithms build data structures that record contributions of light paths and use them for importance sampling. These data structures typically are fitted in a per-frame basis and do not easily extend to animation rendering. In this project, we will explore ways to update the path guiding data structures over time. A potential point of reference is the neural radiance caching work from Muller et al. [34]. A relatively simple starting point is to implement a simple exponential weighted moving average update for the 5D tree data structure of Muller et al. [32]. Can you come up with heuristics to account for occlusion and parallax? What kind of information we can extract from a renderer that can help updating the data structure?

Jointly-learned path guiding and denoising. Current rendering denoisers are usually trained separately from the rendering algorithms (in particular, they are usually trained using a standard path tracer). Can we train the denoiser with the sampling algorithm together to make them compensate for each other? You might want to read Bako's Offline Deep Importance Sampling [3].

Learned multiple importance sampling. While optimal multiple importance sampling [23] is shown to be optimal variance-wise, it has a few drawbacks: 1) it is computationally expensive, 2) it is only optimal under linear combination of existing contributions, and 3) it is only optimal for unbiased rendering – it does not optimize for the Mean Square Error (bias squared plus variance). Therefore, it is tempting to simply learn to do multiple importance sampling. Take a few features as input (e.g., PDFs, roughness, curvature, ambient occlusion), train a parametric function (does not need to be a neural network) to combine your rendering samples. For this project, you don't need to implement in a renderer to start, you can just try it on some test functions.

Neural mutation for Metropolis light transport. Neural networks have been shown to be helpful for importance sampling in a Monte Carlo path tracer [33]. Can we apply them to Metropolis light transport as well? Similar methods have been explored in the machine learning community for Hamiltonian Monte Carlo [26]. In this project, we will investigate the use of neural networks for designing mutation in a Metropolis light transport renderer. A relatively simple starting point is to implement Levy et al.'s method in a Kelemen-style path tracer and see how well it works. The topic is also broadly related to diffusion models.

Learning to build BVHs. Current Bounding Volume Hierarchy constructions are usually based on heuristics (in particular, surface area heuristics). Can we find a better metric through data-driven methods? Find a better BVH construction by training the construction algorithm and its parameters for many scenes to minimize the ray tracing time. You may want to read Aila et al.'s study on surface area heuristics [1].

Multiple-scattering NeRF with neural scattering fields. Neural Radiance Fields [30] and their variants have been revolutionizing 3D computer vision using inverse volume rendering. However, all of the current NeRF variants do not consider multiple-scattering within the volumes. In this project, we will explore the incorporation of multiple scattering in neural fields. To achieve this, we need a neural

representation that can represent a spatially varying, anisotropoic scattering coefficients and phase function. We will also need a way to importance sample both the phase function and transmittance of this neural representation. Alternatively, we can also explore a voxel-based representation (which can have similar performance to a neural representation! [38]) using the SGGX phase function [16] and spherical harmonics for the scattering coefficients. I would recommend starting from known lighting (e.g., an environment map) and single scattering to make the problem easier. I would also recommend to implement this in Slang to leverage its automatic differentiation feature and GPU backends.

Spatially varying neural BSDF for neural signed distance fields. Another popular neural scene representation is neural signed distance fields [4]. However, similar to the situation of NeRF, existing neural signed distance fields do not model the BSDF of the surfaces (they assume all surfaces are pure emitters). In this project, we will explore modeling a spatially varying BSDF for neural signed distance fields. The network architecture can be a 7D coordinate network that takes a 3D position and two 2D directions and outputs the BSDF response. For importance sampling, it might be useful to make the network a normalizing flow [33] conditioned on the position and incoming direction. Similar to above, I would recommend starting from known lighting (e.g., an environment map) and direct lighting to make the problem easier. I would also recommend to implement this in Slang to leverage its automatic differentiation feature and GPU backends.

Transmittance evaluation for NeRF training and inference. In Neural Radiance Fields, the transmittance is typically evaluated using ray marching. This is different from the ones we did in Homework 2 (e.g., ratio tracking). Implement advanced unbiased transmittance evaluation schemes (e.g., Keuttnen et al.'s unbiased ray marcher [19]), and compare with standard NeRF ray marching (in terms of both quality and speed). For training (i.e., derivative evaluation), you might want to implement Nimier-David et al.'s differential tracking [35]. As usual, I would also recommend to implement this in Slang to leverage its automatic differentiation feature and GPU backends.

Faster Monte Carlo PDE solvers. Sawhney et al. [39] recently discussed a type of PDE solver that has very similar computational patterns with rendering. Can you think of a way to make it faster by applying techniques we learned in the course? For example, would something like Metropolis light transport help?

Non-photorealistic rendering. The course so far focuses mostly on physics-inspired models. However, there is a huge space for exploration of non-physical models. How do we define the models? How do we combine with existing renderers? Read my CSE 167 slides and our inverse inverse rendering paper for inspirations.

Faster vector graphics rendering. There are mainly two ways to render vector graphics currently. One is more like ray tracing [12] where we shoot a ray from the pixel sample to see how many intersections it makes with the curves. One is more akin to rasterization [21] where we convert the curves to triangle meshes and apply implicitization to determine whether the point is inside the shape or not. It is unclear which one is faster, or what even is the space of possible design for vector graphics rendering. Do a survey of the algorithms in this space, implement some combination of them, and ideally come up with a fast algorithm for vector graphics rendering. I find Raph Levien's blog post to be very helpful.

Megakernel vs Wavefront. Laine et al. studied two styles of ray tracing on GPU back in 2013 [25]: megakernel style and wavefront style. They concluded that for complex, production material shaders, wavefront rendering is faster because of better memory locality. However, they didn't have hardware ray tracing back then and the hardware architecture is very different now. Is the conclusion still true today? Study different ways to implement path tracing on GPUs and compare their speed on modern GPU architectures.

Multiple-scattering LEAN or LEADR The normal/displacement map filtering methods LEADN/LEADR [36, 10] convert normal map filtering to a microfacet NDF fitting problem. However, the microfacet models they are using do not consider multiple scattering between the microfacets. It seems that it would be straightforward to apply the recent advances in multiple-scattering microfacet BSDFs to these methods. Would

adding multiple-scattering into LEADN/LEADR improve the visual appearance? Do we need a different multiple-scattering model or microfacet configuration?

References

[1] Timo Aila, Tero Karras, and Samuli Laine. On quality metrics of bounding volume hierarchies. In *High Performance Graphics*, pages 101–107, 2013.

[2] Luke Anderson, Tzu-Mao Li, Jaakko Lehtinen, and Frédo Durand. Aether: An embedded domain specific sampling language for Monte Carlo rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 36(4):99:1–99:16, 2017.

[3] Steve Bako, Mark Meyer, Tony DeRose, and Pradeep Sen. Offline deep importance sampling for Monte Carlo path tracing. *Comput. Graph. Forum (Proc. Pacific Graphics)*, 38(7):527–542, 2019.

[4] Sai Bangaru, Michael Gharbi, Tzu-Mao Li, Fujun Luan, Kalyan Sunkavalli, Milos Hasan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Fredo Durand. Differentiable rendering of neural SDFs through reparameterization. In *ACM SIGGRAPH Asia Conference Proceedings*, 2022.

[5] Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. Unbiased warped-area sampling for differentiable rendering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 39(6):245:1–245:18, 2020.

[6] Laurent Belcour and Pascal Barla. A practical extension to microfacet theory for the modeling of varying iridescence. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 36(4):1–14, 2017.

[7] Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 39(4):148, 2020.

[8] Jiawen Chen, Ilya Baran, Frédo Durand, and Wojciech Jarosz. Real-time volumetric shadows using 1D min-max mipmaps. In *Symposium on Interactive 3D Graphics and Games*, pages 39–46, 2011.

[9] Per H. Christensen. An approximate reflectance profile for efficient subsurface scattering. In *ACM SIGGRAPH Talks*, 2015.

[10] Jonathan Dupuy, Eric Heitz, Jean-Claude Iehl, Pierre Poulin, Fabrice Neyret, and Victor Ostromoukhov. Linear efficient antialiased displacement and reflectance mapping. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 32(6):1–11, 2013.

[11] Luis E Gamboa, Adrien Gruson, and Derek Nowrouzezahrai. An efficient transport estimator for complex layered materials. *Comput. Graph. Forum (Proc. Eurographics)*, 39(2):363–371, 2020.

[12] Francisco Ganacim, Rodolfo S. Lima, Luiz Henrique de Figueiredo, and Diego Nehab. Massively-parallel vector graphics. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 33(6):229:1–229:14, 2014.

[13] Pascal Grittmann, Ömercan Yazici, Iliyan Georgiev, and Philipp Slusallek. Efficiency-aware multiple importance sampling for bidirectional rendering algorithms. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 41(4), 2022.

[14] Yu Guo, Miloš Hašan, and Shuang Zhao. Position-free Monte Carlo simulation for arbitrary layered bsdfs. *ACM Transactions on Graphics (ToG)*, 37(6):1–14, 2018.

[15] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 27(5):130:1–130:8, 2008.

[16] Eric Heitz, Jonathan Dupuy, Cyril Crassin, and Carsten Dachsbacher. The sggx microflake distribution. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 34(4):1–11, 2015.

[17] Eric Heitz, Johannes Hanika, Eugene d'Eon, and Carsten Dachsbacher. Multiple-scattering microfacet BSDFs with the Smith model. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 35(4):1–14, 2016.

[18] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. A simple and robust mutation strategy for the Metropolis light transport algorithm. *Comput. Graph. Forum (Proc. Eurographics)*, 21(3):531–540, 2002.

[19] Markus Kettunen, Eugene D'Eon, Jacopo Pantaleoni, and Jan Novák. An unbiased ray-marching transmittance estimator. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 40(4), 2021.

[20] Markus Kettunen, Marco Manzi, Miika Aittala, Jaakko Lehtinen, Frédo Durand, and Matthias Zwicker. Gradient-domain path tracing. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 34(4):123:1–123:13, 2015.

[21] Mark J. Kilgard and Jeff Bolz. GPU-accelerated path rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(6):172:1–172:10, 2012.

[22] Claude Knaus and Matthias Zwicker. Progressive photon mapping: A probabilistic approach. *ACM Trans. Graph.*, 30(3):25:1–25:13, 2011.

[23] Ivo Kondapaneni, Petr Vévoda, Pascal Grittmann, Tomáš Skřivan, Philipp Slusallek, and Jaroslav Křivánek. Optimal multiple importance sampling. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 38(4):1–14, 2019.

[24] Eric P. Lafortune and Yves D. Willems. A 5D tree to reduce the variance of Monte Carlo ray tracing. In *Rendering Techniques (Proc. EGWR)*, pages 11–20, 1995.

[25] Samuli Laine, Tero Karras, and Timo Aila. Megakernels considered harmful: wavefront path tracing on GPUs. In *High Performance Graphics*, pages 137–143, 2013.

[26] Daniel Levy, Matt D. Hoffman, and Jascha Sohl-Dickstein. Generalizing Hamiltonian Monte Carlo with neural networks. In *International Conference on Learning Representations*, 2018.

[27] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable Monte Carlo ray tracing through edge sampling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 37(6):222:1–222:11, 2018.

[28] Daqi Lin, Markus Kettunen, Benedikt Bitterli, Jacopo Pantaleoni, Cem Yuksel, and Chris Wyman. Generalized resampled importance sampling: Foundations of ReSTIR. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 41(4):75:1–75:23, 2022.

[29] Stephen R Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan. Light scattering from human hair fibers. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 22(3):780–791, 2003.

[30] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[31] Zackary Misso, Yining Karl Li, Brent Burley, Daniel Teece, and Wojciech Jarosz. Progressive null-tracking for volumetric rendering. In *SIGGRAPH Conference Proceedings*, 2023.

[32] Thomas Müller, Markus Gross, and Jan Novák. Practical path guiding for efficient light-transport simulation. *Comput. Graph. Forum (Proc. EGSR)*, 36(4):91–100, 2017.

[33] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Trans. Graph.*, 38(5):1–19, 2019.

[34] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. Real-time neural radiance caching for path tracing. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 40(4):36:1–36:16, 2021.

[35] Merlin Nimier-David, Thomas Müller, Alexander Keller, and Wenzel Jakob. Unbiased inverse volume rendering with differential trackers. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 41(4):44:1–44:20, 2022.

[36] Marc Olano and Dan Baker. Lean mapping. In *Symposium on Interactive 3D Graphics and Games*, pages 181–188, 2010.

[37] Radu Alexandru Rosu, Shunsuke Saito, Ziyan Wang, Chenglei Wu, Sven Behnke, and Giljoo Nam. Neural strands: Learning hair geometry and appearance from multi-view images. In *European Conference on Computer Vision*, pages 73–89, 2022.

[38] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Computer Vision and Pattern Recognition*, 2022.

[39] Rohan Sawhney and Keenan Crane. Monte Carlo geometry processing: A grid-free approach to pde-based methods on volumetric domains. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 39(4), 2020.

[40] Theo Thonat, Francois Beaune, Xin Sun, Nathan Carr, and Tamy Boubekeur. Tessellation-free displacement mapping for ray tracing. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 40(6):1–16, 2021.

[41] Ryusuke Villemin and Christophe Hery. Practical illumination from flames. *Journal of Computer Graphics Techniques*, 2(2):142–155, 2013.

[42] Beibei Wang, Wenhua Jin, Jiahui Fan, Jian Yang, Nicolas Holzschuch, and Ling-Qi Yan. Position-free multiple-bounce computations for Smith microfacet BSDFs. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 41(4):134:1–134:14, 2022.

[43] Yu-Chen Wang, Yu-Ting Wu, Tzu-Mao Li, and Yung-Yu Chuang. Learning to cluster for rendering with many lights. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 40(6):1–10, 2021.

[44] Markus Worchel and Marc Alexa. Differentiable rendering of parametric geometry. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 42(6), 2023.

[45] Peiyu Xu, Sai Bangaru, Tzu-Mao Li, and Shuang Zhao. Warped-area reparameterization of differential path integrals. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 42(6), 2023.

[46] Kai Yan, Christoph Lassner, Brian Budge, Zhao Dong, and Shuang Zhao. Efficient estimation of boundary integrals for path-space differentiable rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 41(4):123:1–123:13, 2022.

[47] Ling-Qi Yan, Miloš Hašan, Wenzel Jakob, Jason Lawrence, Steve Marschner, and Ravi Ramamoorthi. Rendering glints on high-resolution normal-mapped specular surfaces. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 33(4), 2014.

[48] Yonghao Yue, Kei Iwasaki, Bing-Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. Unbiased, adaptive stochastic sampling for rendering inhomogeneous participating media. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 29(6), 2010.

[49] Yonghao Yue, Kei Iwasaki, Bing-Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. Toward optimal space partitioning for unbiased, adaptive free path sampling of inhomogeneous participating media. *Comput. Graph. Forum (Proc. Pacific Graphics)*, 30(7):1911–1919, 2011.

[50] Cem Yuksel. Stochastic lightcuts. In *High-Performance Graphics*, pages 27–32, 2019.

[51] Tizian Zeltner, Iliyan Georgiev, and Wenzel Jakob. Specular manifold sampling for rendering high-frequency caustics and glints. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 39(4), 2020.