Differentiable Path Tracing by Regularizing Discontinuities

Peter Quinn
McGill University
peter.quinn@mail.mcgill.ca

Jérôme Parent-Lévesque
Mila, Université de Montréal
jerome.parent-levesque@umontreal.ca

Cengiz Oztireli
University of Cambridge, Google
aco41@cam.ac.uk

Derek Nowrouzezahrai
McGill University
derek@cim.mcgill.ca

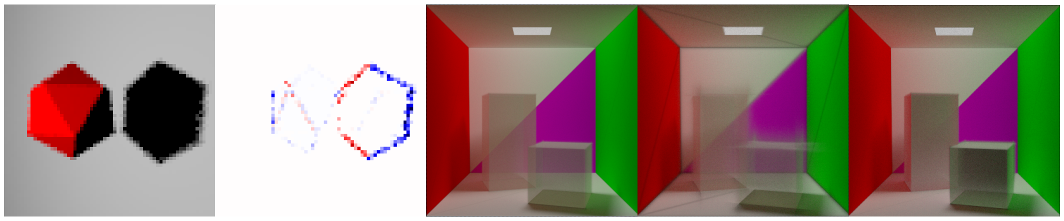

Figure 1: 1b shows the gradients calculated for the icosahedron in 1a for a horizontal translation. 1c shows Cornell box scene with global illumination and Russian roulette termination. 1d has larger σ, resulting in more blur. 1e has smaller γ, resulting in less transparency.

Abstract

Recently, viewing computer vision as an inverse rendering problem has led to a growing interest in differentiable rendering. We introduce a novel differentiable path tracing algorithm where discontinuities in the rendering process are regularized through blurring of the geometry. Our differentiable renderer implements full global illumination and has parameters for controlling the regularization, which allows for some control over the smoothness of the loss landscape. We successfully apply our system to solve several examples of challenging inverse rendering optimization problems that involve more complex light transport scenarios that cannot be handled by rasterization-based differentiable renderers.

1 Introduction

The field of computer vision has seen significant advancements due to the proliferation of machine learning (ML) techniques and specialized neural networks architectures [1] [2]. However, many of these modern techniques focus only on the 2D image, and are agnostic to the underlying 3D world and geometry that are behind the image formation process. Inverse graphics is the process of obtaining a detailed description of a 3D scene from an image. One technique that is useful in inverse graphics is differentiable rendering, which allows for the computation of gradients through the rendering process [3] [4]. These gradients can then be used to optimize estimates of scene parameters to closely match a target image.

In recent years, differentiable rendering has been increasingly used in different ML applications such as mesh reconstruction [5] [6] and lighting reconstruction [7]. The key idea being that incorporating an explicit process converting 3D geometry into 2D images may allow for better generalization.

Differentiable rendering involves modifying the rendering process or adding additional computations to handle discontinuities in the computation of pixel colour, which would otherwise prevent useful gradients from being calculated. There are two main methods for rendering images, rasterization and ray tracing, both of which have been explored for differentiable rendering.

Rasterization uses multiple matrix multiplications accelerated by dedicated hardware (GPU) to efficiently project the geometry in the scene onto the plane defined by the camera position, direction and field of view. This comes at the cost of realism since light paths consisting of multiple bounces are ignored. Several works such as SoftRas [5] [6] and DIB-R [7], and Neural Mesh Renderer [8] have implemented rasterization-based differentiable renderers and applied them to ML applications such as mesh reconstruction. Earlier works look at differentiable rendering for pose estimation [9] [10].

On the other hand, ray tracing produces a detailed, physically accurate image by simulating light rays propagating through a scene. Images produced using ray tracing are able to capture complex lighting effects accurately, at the cost of more computation time. Some differentiable rendering techniques for ray tracing have appeared in the graphics literature in recent years [11] [12] [13] [14] [15] [16], but their high computational costs have limited their use in ML applications.

While the rendering process is trivially differentiable with respect to the material properties when using a simple continuous material model, getting gradients with respect to the scene geometry requires more work. In traditional rendering algorithms, there are essentially two discrete, non differentiable steps that need to be modified in order to make the rendering differentiable with respect to geometry: Discontinuities at the edges of geometry, and determination of the surface closest to the camera.

Contributions In this work, we present a novel formulation for differentiable path tracing based on an extension of the probabilistic view of triangle visibility presented by [5] that models global illumination effects while allowing gradient computation with respect to both geometry and material properties. We implement this formulation in Python using the PyTorch library to allow for simple integration with other common machine learning libraries and pipelines. We apply our technique to several example inverse rendering problems involving complex light transport effects such as shadows, indirect lighting, reflections, and optimization of object and vertex positions. Additionally, we show results on optimizing for camera distribution effects such as depth of field and motion blur. To the best of our knowledge our method is the first to support such effects.

2 Method

We take an approach similar to SoftRas [6] for handling the edge and depth ordering discontinuities, but extend their work to a path tracing setting, with multiple bounces and with support for camera distribution effects. An advantage to our approach compared to other global illumination differentiable renderers (such as [11] [12]) is that it does not require casting additional rays or taking additional samples to calculate gradients.

Edge Discontinuity We view triangles as a probability distribution in 3D space, rather than just in the image plane as proposed in SoftRas. With this approach, a ray can intersect with a triangle in the entire plane of triangle. The probability of intersection with the triangle in the triangle's plane at some point x in 3D space is given by Equation 1:

$$p(x, \sigma) = \text{sigmoid}\left(\text{sign}(d(x))\frac{d(x)^2}{\sigma}\right) \tag{1}$$

where $d(x)$ is the signed distance (positive inside) to the nearest edge of the triangle (in the plane of the triangle) and σ is a value used to control the edges' sharpness. This results in the triangle having the highest probability inside the triangle, $p(x) = .5$ at an edge, and decays asymptotically to 0 as the distance to the edges tends to infinity. $d(x)$ is thus a smooth and differentiable function of the vertex positions of the triangle, which is necessary for the computation of gradients with respect to the vertex positions.

Depth Ordering Discontinuity To handle the discrete operation that results from determining the surface nearest to the camera, we again use a similar approach as taken in SoftRas [6]. We determine the colour of pixel I_j using a weighted sum to blur together the contributions from all surfaces along the path of the ray (Equation 2). The weight W_k of surface k is based on its distance to the camera z_k and the probability of the triangle at the point of intersection $p(x_k, \sigma)$ in its plane (Equation 3). γ controls the weighting of the surfaces. ϵ is a small constant for numerical stability. This weighted sum removes the discrete operation of determining the closest surface and is a form of order-independent transparency [17].

$$I_j = \sum_k W_k \, S(x_k) \qquad (2)$$

$$W_k = \frac{p(x_k, \sigma) \exp\left(z_k/\gamma\right)}{\sum_j p(x_j, \sigma) \exp\left(z_j/\gamma\right) + \exp(\epsilon/\gamma)} \qquad (3)$$

However, computing the shading $S(x_k)$ for every triangle k intersected by a given ray requires casting additional rays to determine the direct and indirect lighting. Doing this for multiple surfaces, for multiple bounces, would result in an exponential increase in the number of rays being cast. This is undesirable as the computation power needed would explode. We propose a numerical method to approximate this sum while using a constant number of rays.

Tracing Subsequent Bounces We use importance sampling [18] to estimate Equation 2. We importance sample one surface based on the weight of its contribution W_k in the sum, and trace additional rays for determining shading only for this surface. Since we pick only a single surface to compute shading for, we are able to keep a constant number of rays per bounce.

Explicit Connections and Shadows To reduce noise, we implement explicit path tracing. For each shading point, an emitter is selected with a uniform probability. A point on the emitter is selected uniformly over its non blurred area. A shadow ray is then traced to determine visibility between the shaded point and the selected point on the emitter. Visibility is computed as *transmission* along the shadow ray: $1 - \prod_k (1 - p(x_k, \sigma))$, for intersections k that occur between the shaded point and the emitter. This factor attenuates the emitted light L_e, before computing the shading using the Bidirectional Reflectance Distribution Function (BRDF). We use a mixture BRDF model that has both diffuse (Lambertian) and specular (Phong) components [19]. Since visibility in these shadow rays is differentiable, this allows computing gradient signals from the shadow of an object.

Textures Textures are stored as H x W x 3 tensor for each mesh. When an intersection point is found, we sample the texture using the UV coordinates of the triangle's vertices to get an albedo value. The values stored in the texture map are naturally differentiable with no special attention needed.To handle areas outside of triangle, the texture is wrapped to cover the extended area.

2.1 Camera Effects

Camera effects can be added to rendered images to simulate features that arise from the physical properties of a camera and lens. We implement two of the most common camera effects differentiably, namely depth of field and motion blur. To our knowledge, ours is the first work to include differentiable camera effects.

Depth of Field Depth of field is an effect that results from the focal length of the lens and the size of the aperture in physical cameras. It results in objects far away from the focal plane being blurred. This effect is simulated by jittering the origin of the rays traced from the camera in the plane perpendicular to the view direction, while ensuring that the rays intersect the focal plane at the same point as if the origin had not been altered. Several of these jittered images are rendered and averaged together to achieve the effect. This averaging operation is naturally differentiable.

Motion Blur Motion blur is an artifact that occurs in images when objects are moving due to the finite shutter speed of the camera. It results in a drawn out blur of the object along the direction of motion. Motion blur is achieved by assigning objects a velocity, sampling the scene at multiple time steps and averaging the resulting rendered images. This averaging operation is naturally differentiable. We limit our implementation to only linear velocities.

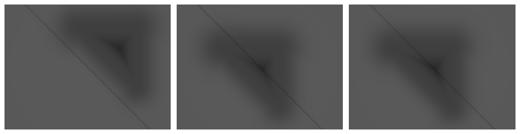
(a) Initial (b) Final (c) Target

Figure 2: The translation of a triangle (not visible in image) is optimized so that its shadow matches the placement of a reference shadow

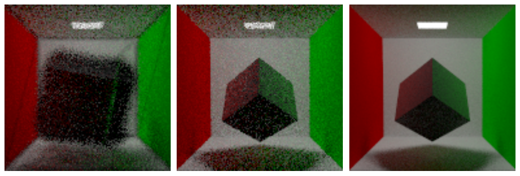
(a) Initial (b) Final (c) Target

Figure 3: The rotation of a white cube is optimized such that the indirect light matches the reference. σ is decayed over time.

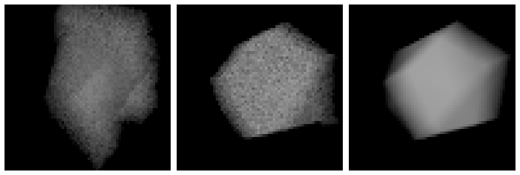
(a) Initial (b) Final (c) Target

Figure 4: The velocity and geometry of an icosahedron are optimized from a coarse initial guess.

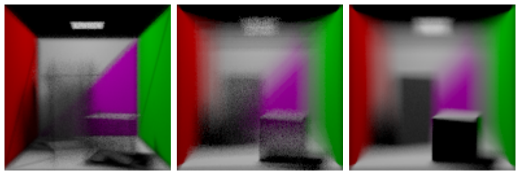
(a) Initial (b) Final (c) Target

Figure 5: The aperture size of the simulated camera is optimized for, causing a depth of field effect.

3 Results

We implemented our differentiable renderer in Python making use of the PyTorch library [20]. Using PyTorch conferred several advantages: parallelization on the GPU by implementing ray tracing as tensor operations, built-in automatic differentiation, optimization tools, and easy integration into Python based machine learning pipelines. For optimization, we used Mean Square Error (MSE) loss and the Adam optimizer [21] available in PyTorch.

During training, we render images with low samples per pixel (SPP) counts, which results in noisy but fast estimates. These renders are then compared to the reference image which is generated with a much higher SPP count. Optimizing based on noisy data has been shown to not be a significant problem as long as there is sufficient data [22], and we find that this holds true in our application. In Figures 1c - 1e, we present example images which show the effects of modifying the edge blurring parameter σ and the depth blurring parameter γ. As γ decreases, the closer surfaces are weighted more heavily. As σ decreases, the blurring of the triangles decreases. We observe that starting with larger values for these parameters and decaying them leads to a more robust optimization. This has the effect of smoothing the gradients earlier on in the optimization and can help avoid local minima.

In Figure 2 and Figure 3, we present examples that demonstrate higher order light transport effects that could not be done with simple rasterization. In Figure 2, the only gradient signal for the translation of the triangle is from the shadow of the triangle. In Figure 3, one bounce indirect light coming from the red and green walls has a significant contribution to the appearance of the white cube. By contrast, rasterization-based differentiable rendering does not typically allow for gradients to be computed with respect to shadows or indirect lighting.

In Figure 4 and Figure 5, we present some optimizations of scenes involving camera effects. In Figure 4, the velocity and geometry of the object are optimized simultaneously. In Figure 5, the aperture size of the camera is adjusted from a small initial value to a large one, which results in an out of focus blur for objects offset from the focal plane.

4 Conclusion and Future Work

We present a novel differentiable approach to path tracing. We apply our approach to solve several inverse rendering problems involving higher order light transport effects. In future work, we would like to improve the both the time and memory efficiency of our differentiable renderer, and increase the number of triangles that can be handled, as these are the current critical limitations. We would also like to further investigate the impact of σ and γ on the characteristics of the loss landscape.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[3] Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*, 2020.

[4] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *European Conference on Computer Vision*, pages 154–169. Springer, 2014.

[5] Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. Soft rasterizer: Differentiable rendering for unsupervised single-view mesh reconstruction. *CoRR*, abs/1901.05567, 2019.

[6] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *CoRR*, abs/1904.01786, 2019.

[7] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Advances in Neural Information Processing Systems*, pages 9609–9619, 2019.

[8] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer, 2017.

[9] Martin de La Gorce, David J Fleet, and Nikos Paragios. Model-based 3d hand pose estimation from monocular video. *IEEE transactions on pattern analysis and machine intelligence*, 33(9):1793–1805, 2011.

[10] Helge Rhodin, Nadia Robertini, Christian Richardt, Hans-Peter Seidel, and Christian Theobalt. A versatile scene model with differentiable visibility applied to generative pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 765–773, 2015.

[11] 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.

[12] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 38(6), November 2019.

[13] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Transactions on Graphics*, December 2019.

[14] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. Radiative backpropagation: an adjoint method for lightning-fast differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(4):146–1, 2020.

[15] Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, and Shuang Zhao. Path-space differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(4):143–1, 2020.

[16] Shuang Zhao, Wenzel Jakob, and Tzu-Mao Li. Physics-based differentiable rendering: From theory to implementation. In *ACM SIGGRAPH 2020 Courses*, SIGGRAPH 2020, New York, NY, USA, 2020. Association for Computing Machinery.

[17] Morgan McGuire and Louis Bavoil. Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques (JCGT)*, 2(2):122–141, December 2013.

[18] Peter W Glynn and Donald L Iglehart. Importance sampling for stochastic simulations. *Management science*, 35(11):1367–1392, 1989.

[19] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2016.

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[22] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*, 2018.