Differentiable Monte Carlo Ray Tracing through Edge Sampling

Proseminar Differentiable Programming

Clemens Dautermann

July 26, 2023

***Abstract*—Differentiable Programming is a technique frequently used to solve optimization problems by minimizing some kind of error function. This requires the error function to be differentiable with respect to the parameters that are to be optimized, a condition which usually does not hold with ray tracing. This report will explain why this problem occurs and present the method developed in [2] to tackle it.**

I. Introduction

One of the main tasks in Computer Graphics is image synthesis. This means "given a 3D scene, output an image depicting the scene". Often it is required for the image to be as realistic as possible, meaning as close to a picture of the scene as if it was set in the real world. This is most commonly achieved using the ray tracing algorithm and is a well studied problem. Doing this in a differentiable way, however, is much less trivial. This stems from the fact that the rendering integral (equation 1) is not differentiable in certain well defined places. This report will illustrate why this is, how it can be mitigated and what might be interesting applications of differentiable ray tracing.

II. Ray Tracing

To formalize the problem of photo realistic image synthesis, an equation has been proposed by Kajiya in 1986 [1]. This equation captures physical light transport for a scene and if solved yields the color for a given point in the scene accounting for most physical light transport phenomena.

$$I(x, x') = g(x, x') \left[\epsilon(x, x') + \int_S \rho(x, x', x'') I(x', x'') dx'' \right]$$

Equation 1: The rendering equation capturing physical light transport. It assigns a value to the intensity of light transported from a point x to a point x'. The geometry term g will be discussed later. The term ϵ accounts for the emissivity of the material at point x. The integral term represents all light scattered from any other point in the scene towards the point x. The integral domain S contains all points in the scene.

This equation (equation 1) is now widely recognized as "the rendering integral". It can not be solved analytically and is thus most commonly approximated using Monte-Carlo integration - i.e. ray tracing. Ray tracing works by backtracking light rays from the light sources in the scene and thus simulates physically realistic lighting.
To do this, rays are cast from the camera, through each pixel in the camera frustum. The intersection point with the scene geometry x is calculated for each ray and material properties (e.g. color, emissivity etc.) are taken into account to calculate the pixel color. From this point more rays are drawn towards each light source. If the light source is visible, its light contributes to the pixel color as well. To account for indirect lighting, the ray "bounces around', yielding a color for some of the points scattering lights towards x. This approximates the integral term in equation 1.
Differentiable ray tracing is the task of calculating the gradient of this process with respect to *any* scene parameter.

III. Problems with differentiability

The geometry term $g(x, x')$ in equation 1 is the main problem when it comes to differentiating the rendering integral. This term is 1 iff x is visible from x' and 0 otherwise.

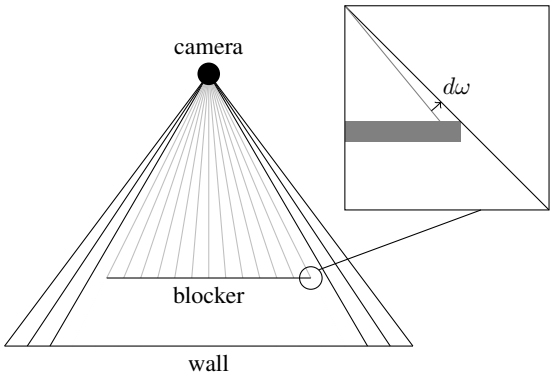
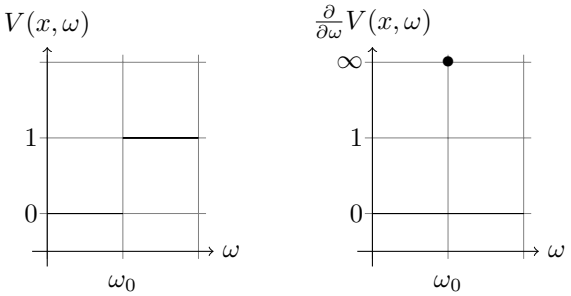


Figure 1: A simple occlusion scenario. An infinitesimal change in ray angle $d\omega$ is sufficient to determine whether the wall is visible or not.

As illustrated in figure 1, an infinitesimal angle change $d\omega$ can lead to the blocker obstructing the wall or the wall being visible. The geometry term is thus a Heaviside step function which when differentiated yields a Dirac delta functional (see figure 2). Since the Dirac delta functional only differs from 0 in ω_0 one point, the probability of sampling it when using uniformly distributed Monte-Carlo integration is 0.



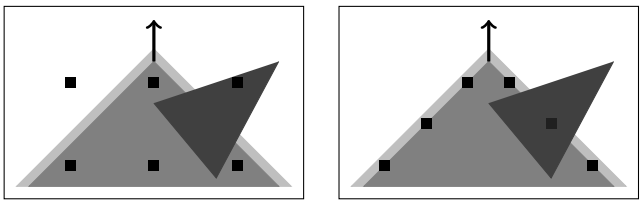
(a) Visibility of a point x with respect to ω

(b) Differentiation of the left graph with respect to ω

Figure 2: The visibility of a point and the differentiation

IV. Proposed solution: Edge sampling

The key observation is that these discontinuities only occur at the edges of meshes. Thus the problem can be mitigated by not sampling a pixel uniformly, but importance sampling the edges of a mesh, hence capturing the Dirac delta spikes.



(a) area sampling (b) edge sampling

Figure 3: The figure depicts the sampling situation in a single pixel. The gradient with respect to the gray triangle moving up is the metric to be calculated. Area sampling does not account for this color change whereas edge sampling does. Primary occlusion is handled correctly.

Consider a situation as depicted in figure 3. When the gray triangle moves up, the ratio of the white area decreases and the ratio of the gray area increases. Thus white should contribute less to the pixel color. Area sampling does not account for this change, since all samples land on an area with the same color before and after movement. Edge sampling does account for this change because especially areas of discontinuity i.e. the edges are sampled. The samples will have a different colors after the triangle moves up, correctly capturing the gradient. Primary occlusion is handled correctly because the sample that intersected the blocker before, will also intersect the blocker after movement as it is independent of the triangle movement.

A. Half spaces

To understand why edge sampling correctly captures the gradient, an explanation of what exactly happens at the point where a ray intersects an edge is required.

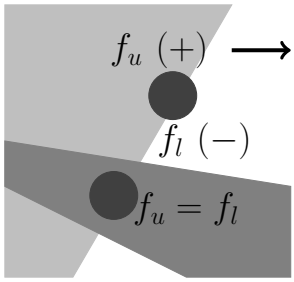
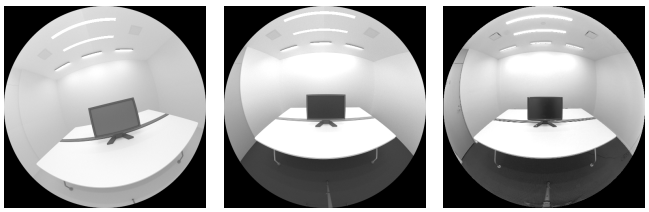


Figure 4: An edge separates the domain into two regions, whose contribution can be considered separately. The contribution of the upper half space increases while the contribution of the lower half space decreases with respect to the gray triangle moving right. Occlusion is handled correctly because there the contribution of either half space is equal.

As seen in figure 4, the edge separates the space into two half spaces. These two halfspaces can now be considered separately and a gradient can be calculated from the color difference. It is also evident that occlusion is handled correctly. This arises from the fact that where occlusion occurs, the half spaces' contribution to color is equal. From that it follows that the gradient will be 0 where occlusion occurs.
This allows the integral to be separated into two parts: one part that represents the discontinuous edges and one part that accounts for the original pixel integral over continuous regions.

V. Results

The main application of differentiable ray tracing discussed in the paper is inverse rendering. That is: Given images of a scene, synthesize a 3D representation of that scene. To do this an error function $e : \Phi \mapsto \mathbb{R}$, that measures how close an approximate render is to a target image with respect to a set of scene parameters Φ, is required.
Given this function and a set of target images the inverse rendering problem can be solved using differentiable ray tracing. The first step is to render an approximation of the scene differentiably. The resulting images are then compared to the target images by calculating the error. Since the rendering process is differentiable the gradient of the error function can be calculated, allowing for a minimization of the error function using a process such as stochastic gradient descent.



(a) initial guess (b) optimized result (c) target

Figure 5: A generic example for how differentiable ray tracing can be used to approximate a solution for the inverse rendering problem.

An example for how this might look is given in figure 5. The optimized image (figure 5, image (b)) has been generated by the authors using differentiable ray tracing and an ADAM gradient descent optimizer. From this image it is evident that the algorithm can handle a multitude of rendering phenomena, such as e.g. primary occlusion, emission and glossy receivers.

References

[1] James T. Kajiya. "The Rendering Equation". In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '86. New York, NY, USA: Association for Computing Machinery, 1986, pp. 143–150. ISBN: 0897911962. DOI: 10.1145/15922.15902. URL: https://doi.org/10.1145/15922.15902.

[2] Tzu-Mao Li et al. "Differentiable Monte Carlo Ray Tracing through Edge Sampling". In: *ACM Trans. Graph.* 37.6 (2018). ISSN: 0730-0301. DOI: 10.1145/3272127.3275109. URL: https://doi.org/10.1145/3272127.3275109.