Physics-based Differentiable Rendering

Differentiable Monte Carlo Ray Tracing through Edge Sampling

TZU-MAO LI, MIT CSAIL
MIIKA AITTALA, MIT CSAIL
FRÉDO DURAND, MIT CSAIL
JAAKKO LEHTINEN, Aalto University & NVIDIA

Date: 10 Mar. 2021
Jingkang Wang
Differentiable Rendering is Important!

- The ability of calculating gradients are crucial to optimization
  - (a) inverse problems, (b) deep learning

---

The diagram illustrates a neural network which takes in a 3D scene with cameras, geometry, lights, and material, and outputs an image. The process is represented as a function $\Phi$. The inverse rendering process is shown as well.
Differentiable Rendering is Important!

- Render and compare approach

update

3D scene: triangle positions camera pose materials ...

Gradients?

image

target

\( \nabla \text{loss} \)

Optimize scenes via differentiable renderer

source

target

3D scene
distance to target
Differentiable Rendering is Challenging!

- Computing the gradient of rendering is **challenging**

Rendering integral includes visibility terms that are not differentiable

\[ I = \iint \text{Pixel filter} \cdot \text{Radiance (another integral)} \, dx \, dy \]

Scene function:

\[ f(x, y; \Phi) = k(x, y)L(x, y) \]

\[ \nabla I = \nabla \iint f(x, y; \Phi) \, dx \, dy \]
Differentiable Rendering is Challenging!

\[ I = \iiint \quad \quad E = \frac{1}{N} \sum \quad \text{Monte Carlo samples} \]

**Differentiable integrand: Yes**

\[ \left( \frac{\partial}{\partial p_i} \int = \int \frac{\partial}{\partial p_i} \right) \]

Easy to compute (e.g., automatic differentiation)

**Non-differentiable integrand: No**

\[ \left( \frac{\partial}{\partial p_i} \int \neq \int \frac{\partial}{\partial p_i} \right) \]

Can we just use \( \frac{\partial E}{\partial p_i} \) to estimate \( \frac{\partial I}{\partial p_i} \)?
Differentiable Rendering is Challenging!
Issues with Automatic Differentiation
Issues with Automatic Differentiation
Related Work

**Rasterization**

- **OpenDR: an Approximate Differentiable Renderer**
  Matthew Loper, et al.

- **Soft Rasterizer: Differentiable Rendering for Unsupervised Single-View Mesh Reconstruction**
  Shichen Liu, et al.

**Physically based rendering**

- **Differentiable Monte Carlo Ray Tracing Through Edge Sampling**
  Tzu-Mao Li, et al.

- **Mitsuba 2: a Retargetable Forward and Inverse Renderer**
  Merlin Nimier-David, et al.

**Neural rendering**

- **Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations**
  Vincent Sitzmann, et al.

- **BlockGAN: Learning 3D Object-aware Scene Representations from Unlabelled Images**
  Thu Nguyen-Phuoc, et al.
Contributions

- This paper proposed a general physically-based differentiable render

glossy reflection  mirror reflection  shadow  global illumination
Contributions

- This paper proposes a **general physically-based differentiable renderer**
  - **General differentiable path tracer**
    - a stochastic approach based on **Monte Carlo** ray tracing to estimate both the integral and the gradients of the pixel filter’s integral
  - **Handling geometric discontinuities**
    - a combination of standard area sampling and novel **edge sampling** to deal with smooth and discontinuous regions

- This paper shows
  - The utility of proposed differentiable renderer in several applications (inverse rendering, 3D adversarial examples)
  - Better performance than two previous differentiable renderers (OpenDR & Neural Mesh Rendering)
Physically-based Rendering

- The Rendering Equation
The Rendering Equation

\[ L_o(X, \hat{\omega}_o) = L_e(X, \hat{\omega}_o) + \int_{S^2} L_i(X, \hat{\omega}_i) f_X(\hat{\omega}_i, \hat{\omega}_o) |\hat{\omega}_i \cdot \hat{n}| \, d\hat{\omega}_i \]

Outgoing direction

A point in the scene

All incoming directions (a sphere)

Incoming direction

Surface normal

Credit: https://news.developer.nvidia.com/ray-tracing-essentials-part-6-the-rendering-equation/
The Rendering Equation

\[ L_o(X, \hat{ω}_o) = L_e(X, \hat{ω}_o) + \int_{S^2} L_i(X, \hat{ω}_i) f_X(\hat{ω}_i, \hat{ω}_o) |\hat{ω}_i \cdot \hat{n}| d\hat{ω}_i \]

Credit: https://news.developer.nvidia.com/ray-tracing-essentials-part-6-the-rendering-equation/
Rendering = Sampling

color change when blue triangle moves up?
Key idea: Explicitly integrate the boundaries
Mathematical formulation

- Model the edge as the step function
- Each pixel is an integral over the step functions

\[ \nabla \int s(x) \, dx = \int \nabla s(x) \, dx \]

Step function

Dirac delta

\[ \delta(x) \]
Mathematical formulation

• A smooth shading function $f$ multiples to the step function $s$

$$\nabla (s \cdot f) = (\nabla s) \cdot f + s \cdot (\nabla f)$$
1D Derivatives

\[ \int_{x=0}^{x=1} \text{(the blue area)} \]

\[ x < p \ ? 1 : 0.5 \]
1D Derivatives

derivative w.r.t. p = this purple infinitesimal area (0.5 dp)

\[ \int_{x=0}^{x=1} (\text{the blue area}) \]

\[ x < p \quad ? \quad 1 : 0.5 \]
• Trick: move the discontinuities to the integral boundaries

\[
\int_{x=0}^{x=1} \begin{cases} 
1 & \text{if } x < p \\
0.5 & \text{if } x = p
\end{cases}
\]

= \int_{x=0}^{x=p} 1 + \int_{x=p}^{x=1} 0.5
1D Derivatives

\[
\frac{\partial}{\partial p} \left( \int_{x=0}^{x=p} 1 + \int_{x=p}^{x=1} 0.5 \right) = 1 - 0.5
\]

\[
\int_{x=0}^{x=1} x < p \ ? 1 : 0.5
\]

(derivative of blue area w.r.t. \( p \))
Discontinuity derivatives = differences at discontinuities

\[
\frac{\partial}{\partial p} \int f \, dp = \int \frac{\partial}{\partial p} f \, dp + \sum (f_+ - f_-)
\]

"the Leibniz's integral rule"
Discontinuity derivatives = differences at discontinuities

\[ \frac{\partial}{\partial p} \int \bigg( \frac{\partial}{\partial p} \bigg) = \int \frac{\partial}{\partial p} + \sum (f_- - f_+) \]

"the Leibniz's integral rule"
\[ \frac{\partial}{\partial p} \int \int \text{ } = \int \int \frac{\partial}{\partial p} \text{ } + \int \text{ } \]

Reynolds transport theorem
[Reynolds 1903]

interior derivative
boundary derivative
Mathematical formulation

- Scene function \( f(x, y; \Phi) \)

- Pixel Color \( I = \iint f(x, y; \Phi)dx\,dy \)

- Gradient \( \nabla I = \nabla \iint f(x, y; \Phi)dx\,dy \)

- All discontinuities happen in the scene edges

\[
f(x, y; \Phi) = \theta(\alpha(x, y))f_u(x, y; \Phi) + \theta(-\alpha(x, y))f_l(x, y; \Phi)
\]

\[
I = \iint f(x, y; \Phi)dx\,dy = \sum_i \iint \theta(\alpha_i(x, y))f_i(x, y; \Phi)dx\,dy
\]

\[
\alpha(x, y) = Ax + By + C
\]
Mathematical formulation

- Using the Chain rule

\[ \nabla \int \theta(\alpha(x, y)) f(x, y; \Phi) dx dy = \int \delta(\alpha(x, y)) \nabla \alpha(x, y) f(x, y; \Phi) dx dy + \int \nabla f(x, y; \Phi) \theta(\alpha(x, y)) dx dy \]

Edge sampling

Area sampling
Generalization & Scalability

- Generalizable to shadow & interreflection
- Use importance sampling to sample edges and pick points (Hill and Heitz 2017)

area of a light source

select an edge & pick a point
Algorithms

dPT\( (x, \omega_o) \): # Estimate \( L(x, \omega_o) \) and \( \frac{d}{d\pi}[L(x, \omega_o)] \) jointly

sample \( \omega_{i,1} \in S^2 \) with probability \( p_{i,1} \)
\( y \leftarrow \text{rayIntersect}(x, \omega_{i,1}) \)
(\( L_i, \hat{L}_i \) \( \leftarrow \) dPT(\( y, -\omega_{i,1} \))

\[
L \leftarrow \frac{f_s(x, \omega_{i,1}, \omega_o) L_i}{p_{i,1}}
\]
\[
\hat{L} \leftarrow \frac{\frac{d}{d\pi}[f_s(x, \omega_{i,1}, \omega_o)] L_i + f_s(x, \omega_{i,1}, \omega_o) \hat{L}_i}{p_{i,1}}
\]

sample \( \omega_{i,2} \in \partial S^2 \) with probability \( p_{i,2} \)
\( \hat{L} \leftarrow \hat{L} + \frac{V_{\partial S^2}(x, \omega_{i,2}) f_s(x, \omega_{i,2}, \omega_o) \Delta L_i(x, \omega_{i,2})}{p_{i,2}} \)

return \( (L + L_e(x, \omega_o), \hat{L} + \frac{d}{d\pi} L_e(x, \omega_o)) \)

Rendering equation

\[
L(\omega_o) = \int_{S^2} \frac{f_{RE}(\omega_i)}{f_s(\omega_i, \omega_o) L_i(\omega_i)} d\sigma(\omega_i) + L_e(\omega_o)
\]

Differential rendering equation

\[
\frac{d}{d\pi} L(\omega_o) = \int_{S^2} \frac{d}{d\pi} f_{RE}(\omega_i) d\sigma(\omega_i)
\]

+ \[
\frac{d}{d\pi} L_e(\omega_o)
\]

Standard PT w/ symbolic differentiation

Monte Carlo edge sampling
Algorithms

dPT(\(x, \omega_o\)): # Estimate \(L(x, \omega_o)\) and \(\frac{d}{d\pi}[L(x, \omega_o)]\) jointly

- sample \(\omega_{i,1} \in S^2\) with probability \(p_{i,1}\)
- \(y \leftarrow \text{rayIntersect}(x, \omega_{i,1})\)
- \((L_i, \hat{L}_i) \leftarrow \text{dPT}(y, -\omega_{i,1})\)
- \(L \leftarrow \frac{f_s(x, \omega_{i,1}, \omega_o) \cdot L_i}{p_{i,1}}\)
- \(\hat{L} \leftarrow \frac{\frac{d}{d\pi}[f_s(x, \omega_{i,1}, \omega_o)] \cdot L_i + f_s(x, \omega_{i,1}, \omega_o) \cdot \hat{L}_i}{p_{i,1}}\)

- sample \(\omega_{i,2} \in \partial S^2\) with probability \(p_{i,2}\)
- \(\dot{L} \leftarrow \hat{L} + \frac{V_{\partial S^2}(x, \omega_{i,2}) \cdot f_s(x, \omega_{i,2}, \omega_o) \cdot \Delta L_i(x, \omega_{i,2})}{p_{i,2}}\)

return \((L + L_e(x, \omega_o), \dot{L} + \frac{d}{d\pi}L_e(x, \omega_o))\)
Experiments – Synthetic examples

- Optimizing 6 triangle vertices
Experiments – Synthetic examples

- Optimizing blocker vertices

Source

Target
Experiments – Synthetic examples

camera & teapot material  
Target

logo translation

Source

camera
Experiments – Synthetic examples

- Compare with central finite differences (32 x 32 scenes)
Experiments – Synthetic examples

- Sampling with or without edge importance sampling
Experiments – Inverse rendering

- Optimizing camera pose, light emission and materials
Experiments – Inverse rendering

- Optimizing camera pose, light emission and materials
Experiments – Inverse rendering

- Optimizing camera pose, light emission and materials

camera gradient  table albedo gradient  light gradient
Experiments – 3D adversarial examples

- Optimizing vertex position, camera pose, light intensity, position

VGG 16:
53% street sign
6.7% handrail

5 iterations:
26.8% handrail
20.2% street sign

25 iterations:
23.3% handrail
3.4% street sign
Limitations

- **Performance** (rendering speed & large variance):
  - Edge sampling and auto differentiation are slow (bottleneck)
  - It is a challenging task to find all object edges and sampling them

- **Assumptions:**
  - Interpenetrating geometries
  - parallel edges (non-differentiable)
  - Surface only light transport
Contributions (recap)

- Previous works
  - Differentiable rendering that targets specific cases (faces, hands, etc.) => hard to generalize
  - Fast, approximate general renderers (OpenDR, Neural Mesh Rendering) => simplified models
  - challenges: estimating the derivative corresponding to the integral of the rendering equation

- This paper proposes a general physically-based differentiable renderer
  - General differentiable path tracer
  - Handling geometric discontinuities

- This paper shows
  - The utility of proposed differentiable renderer in several applications (inverse rendering, 3D adversarial examples)
  - Better performance than two previously proposed differentiable renderers
Follow-up works

- Addressing the discontinuity problem in the rendering equation

Handle volumetric light transport (Zhang et al., 2019)

Re-parameterize the integral (Loubet et al., 2019)
Radiative transfer equation (RTE) in operator form

\[ L = K_T K_C L + Q \]

A Differential Theory of Radiative Transfer

Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulakes, Ravi Ramamoorthi, Shuang Zhao

SIGGRAPH Asia 2019
Challenges

• Complex scenes
  - Discontinuity points (i.e., $\partial S^2$) can be expensive to detect

• Scaling out to millions of parameters
Reparameterizing Discontinuous Integrands for Differentiable Rendering

A scene with complex geometry and visibility (1.8M triangles)

Gradients with respect to scene parameters that affect visibility
Key Idea: Re-parameterizing Integrals

\[ I = \int k(x) \mathbb{1}_{x>p} \, dx \]

\[ \frac{\partial I}{\partial p} = ? \]

Change of variable:

\[ X = x - p \]

\[ I = \int k(X + p) \mathbb{1}_{X>0} \, dx \]

- Same value of the integral
- Same sample positions
- Different partial derivatives for MC samples
Key Idea: Re-parameterizing Integrals

Non-differentiable Monte Carlo estimates

Differentiable Monte Carlo estimates

Pixel filter or BRDF
Integrals with Large Support

No useful reparameterization

Simple changes of variables make estimates differentiable
(assuming: infinitesimal translation)
Integrals with Large Support

- Estimating the same integral with a different sampling technique
Integrals with Large Support

Assumption (Small angular support):
Removal of discontinuities using rotations

\[ I = \int_{S^2} f(\omega, \theta) \, d\omega = \int_{S^2} f(R(\omega, \theta), \theta) \, d\omega \]

\[ E = \frac{1}{N} \sum \frac{f(R(\omega_i, \theta), \theta)}{p(\omega_i, \theta)} \approx I \]

Handling with large support

\[ \int_{S^2} f(\omega) \, d\omega = \int_{S^2} \int_{S^2} f(\mu) k(\mu, \omega) \, d\mu \, d\omega, \int_{S^2} k(\mu, \omega) \, d\mu = 1. \quad \forall \omega \in S^2 \]

\[ I \approx E = \frac{1}{N} \sum \frac{f(R_i(\mu_i, \theta), \theta) \, k(R_i(\mu_i, \theta), \omega_i(\theta), \theta)}{p(\omega_i(\theta), \theta) \, p_k(\mu_i)} \]

(a) Differentiable rotation of directions

(b) Notations for our spherical convolutions
Results
Results

Glossy reflection

Shadows

Refraction

Ours

Reference (Finite differences)

Without changes of variables
Results

Glossy reflection

Mesh subdivision

Edge sampling [Li et al. 2018]
Reparameterization
Reference
Finite differences
Challenges

Scene parameters $x \in \mathcal{X}$

Reverse-mode AD

Gradients
- Roughness $\alpha$
- Diffuse color
- Diffuse texture

Rendering algorithm
Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering

Merlin Nimier-David, Sébastien Speierer, Benoit Ruïz, Wenzel Jakob

SIGGRAPH 2020
Radiative Backpropagation

Normal rendering
- Transporting from sensor/light may yield lower variance.

Differentiable rendering
- Transporting from objects is completely impractical.
Render and Compare

\[ g(\text{Rendering}) = \| \text{Rendering} - \text{Target} \|^2 \]

The problem: \( \min_{x \in \mathcal{X}} g(f(x)) \)

\[ z = g(f(x)) \]

We need: \( \frac{\partial z}{\partial x} \)
Render and Compare

Objective function

\[
\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}
\]

Rendering

Scene parameters

\[
\frac{\partial z}{\partial y} = \frac{\partial z}{\partial x}
\]

Sensitivity gradients
Chain Rule

\[
\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \left(\ldots\right) \cdot \frac{\partial f_s(x, \omega, \omega')}{\partial x}
\]
Chain Rule

\[
\frac{\partial z}{\partial x} = \left( \ldots \right) \cdot \frac{\partial f_s(x, \omega, \omega')}{\partial x}
\]

"Derivative shader"
Easy & self-contained
Pipeline Overview

- Scene parameters $x \in \mathcal{X}$
- Primal rendering
- Reference image
- Objective function
- $g(y)$
- Backpropagate
- Adjoint rendering
- Gradient of objective w.r.t. each pixel $L(y, \omega')$
- Radiative backpropagation
- Roughness $\alpha$
- Diffuse color
- Diffuse texture

$\delta y$
Radiative Backpropagation
Surface Texture Optimization
Surface Texture Optimization
Volume Density Optimization

Equal time (2.5 min)

Reference
Initial state
Ours (biased I)
Autodiff-based
Ours (biased II)
Ours
Challenges Remain

Complex light transport

Complex geometry & motion
Follow-up works

- Estimate the derivatives of the path integral formulation

Path space differentiable rendering (Zhang et al., 2020)
Path Integral for Forward Rendering

\[ I = \int_{\Omega} f(\mathbf{x}) \, d\mu(\mathbf{x}) \]

- Introduced by Veach [1997]
- Foundation of sophisticated Monte Carlo algorithms (e.g., BDPT, MCMC rendering)

Light path \( \mathbf{x} = (x_0, x_1, x_2, x_3) \)
• Differential path integral
  – Separated interior and boundary components
  \[ \frac{d}{d\pi} \int_{\Omega} f \, d\mu = \int_{\Omega} \frac{df}{d\pi} \, d\mu + \int_{\partial\Omega} g \, d\mu' \]

• Reparameterization
  – Only need to consider silhouette edges

• Unbiased Monte Carlo methods
  – Unidirectional and bidirectional algorithms
  – No silhouette detection is needed
**Parameter:** rotation angle of the object

**Equal-sample comparison**

- Path tracing w/ edge sampling [Li et al. 2018, Zhang et al. 2019]
- Reparameterization [Loubet et al. 2019]
- Path-space, unidir. [Zhang et al. 2020]
Summary

- **Differentiable rendering is challenging**
  - Discontinuities are everywhere
  - Automatic-differentiation is time & space consuming

- **Physics-based differentiable rendering**
  - Dealing with the discontinuities:
    - Edge sampling (Li et al. 2018, Zhang et al. 2019)
    - Reparameterization (Loubet et al. 2019)
    - Path integral formulation (Zhang et al. 2020)

  - Dealing with memory issue:
    - Radiative Backprop (Nimier-David et al. 2020)
References

- Differentiable Monte Carlo Ray Tracing through Edge Sampling. Li et al., 2018.
- Slides for “Differentiable Monte Carlo Ray Tracing through Edge Sampling”. Li et al., 2018.
- https://rgl.epfl.ch/publications/Loubet2019Reparameterizing
- https://shuangz.com/courses/pbdr-course-sg20/