Differentiable Monte Carlo Ray Tracing through Edge Sampling

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

Date: 16 Feb. 2021
Presenter: Jingkang Wang
Instructor: Animesh Garg
Differentiable Rendering is Important!

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

![Diagram of neural network, 3D scene, image, and inverse rendering process]
Differentiable Rendering is Important!

- Render and compare approach

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

Gradients?

Optimize scenes via differentiable renderer

image

∇ loss

distance to target

source

target

3D scene
Differentiable Rendering is Important!

- Computing the gradient of rendering is challenging!

Rendering integral includes visibility terms that are not differentiable

\[
I = \iiint k(x, y)L(x, y)dx\,dy
\]

Pixel filter \quad Radiance (another integral)

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

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

- **Challenge**: both primary and secondary visibility matter
Contributions

- 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

Specific cases
(Blanz et al. 1999, Gorce et al. 2008, Gkioulekas et al., 2013)

Limited general renders
(Loper and Black 2014, Kato et al. 2018)
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
Physically-based Rendering

- The Rendering Equation
The Rendering Equation

Outgoing direction

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

A point in the scene

All incoming directions (a sphere)

Incoming direction

Surface normal

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 light  Emitted light  Incoming light  Material  Lambert

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: Edge sampling

color change when blue triangle moves up?
Mathematical formulation

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

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

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

$$\nabla (s \cdot f) = (\nabla s) \cdot f + s \cdot (\nabla f)$$

- Dirac delta
- Shading derivatives
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
Mathematical formulation

- Using the Chain rule

\[ \nabla \int \theta(\alpha(x, y)) f(x, y; \Phi) dx dy = \int_{\alpha(x,y)=0} \frac{\nabla \alpha_i(x, y)}{\|\nabla_{x,y} \alpha_i(x, y)\|} f_i(x, y) d\sigma(x,y) + \int \nabla f(x, y; \Phi) \theta(\alpha(x, y)) dx dy \]

\( \text{Edge sampling} \)

\( \text{Area sampling} \)
Generalization & Scalability

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

- Optimizing 6 triangle vertices
Experiments – Synthetic examples

- Optimizing blocker vertices

Source

Target
Experiments – Synthetic examples

Target

Source

camera & teapot material

logo translation

camera
Experiments – Synthetic examples

- Compare with central finite differences (32 x 32 scenes)

(a) triangles  (b) shadow  (c) teapot
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

optimization  target
Experiments – Inverse rendering

- Optimizing camera pose, light emission and materials

camera gradient  table albedo gradient  light gradient
Experiments – 3D adversarialial 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
  - A few hundreds of milliseconds to generate a small image (256x256) with a small number of samples (4)

- **Assumptions:**
  - Mesh
  - Interpenetrating geometries and parallel edges
  - Shader discontinuities
  - Motion blur
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)
Follow-up works

- Estimate the derivatives of the path integral formulation

Path space differentiable rendering (Zhang et al., 2020)
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
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.