Optimizing 3D Scene Reconstruction with Differentiable Ray Tracing

***Abstract*—Traditional rendering techniques struggle with optimizing scene parameters due to their non-differentiable nature. This study introduces a novel approach to scene reconstruction using a differentiable ray tracing framework, enabling gradient-based optimization for refining object properties, lighting, and camera parameters. By leveraging automatic differentiation, the proposed method efficiently reconstructs complex scenes with improved accuracy over conventional optimization techniques. Experimental results demonstrate the effectiveness of this approach in minimizing reconstruction error while maintaining computational efficiency. These findings contribute to the advancement of neural rendering and inverse graphics applications.**

I. Introduction

Rendering transforms three-dimensional scenes into two-dimensional visual representations, crucial for both photorealistic and stylized outputs. Among various techniques, ray tracing [1] offers superior realism by simulating the physical behavior of light, while rasterization [2] remains the method of choice for real-time applications due to its efficiency.

Ray tracing operates by tracing rays from the viewpoint into the scene, computing light interactions such as reflection and refraction. Though visually compelling, this process demands substantial computational resources, making real-time applications difficult.

An extension of the rendering pipeline is the concept of *inverse rendering*, where the objective is to deduce scene parameters—including geometry, material, and lighting—from a given image. A key challenge in implementing inverse rendering lies in the computation of derivatives for each parameter, traditionally requiring error-prone analytical derivations. To address this, *redner* [3] introduced one of the earliest frameworks with analytical gradient support, though it remains limited by the intricacies of symbolic differentiation.

Our approach circumvents these constraints by employing automatic differentiation (AD), specifically source-to-source AD via Zygote [4], built in Julia [5]. Most existing rendering engines are implemented in low-level languages like C++ which lack integration with modern AD tools such as JAX [6] and CasADi [7], creating barriers to interoperability with machine learning ecosystems.

This paper introduces *RayTracer.jl*, a Julia-based, fully differentiable rendering engine that utilizes Zygote for AD, enabling the computation of gradients across arbitrary scene parameters without analytic derivatives. With seamless integration into machine learning frameworks like Flux [8], RayTracer.jl supports efficient inverse rendering pipelines for research and experimentation.

II. Differentiable Ray Tracing

Photorealistic renderers encode intricate lighting and material models, but most are non-differentiable, limiting their use in gradient-based optimization. Our work tackles this by constructing a renderer from scratch in Julia, enabling differentiability by design.

RayTracer.jl leverages Zygote's source-to-source AD capabilities to compute gradients with respect to any scene parameter, such as light positions, object transformations, and material properties. This tight coupling with AD allows integration with Flux and other ML libraries that share the same AD backend.

To enable differentiability, we simplify the ray sampling process—tracing one ray per pixel. Although traditional ray tracing benefits from sampling multiple rays per pixel for antialiasing, this complexity introduces nondifferentiable control flow. Instead, a single-ray-per-pixel model ensures each ray intersects only one triangle, allowing the final color to be expressed as a smooth function of intersection points and material properties (e.g., via barycentric coordinates).

Some limitations persist, notably when rays intersect edges shared by multiple triangles. These points induce non-differentiable behavior, a known issue discussed further in Section VI.

III. Scene Rendering

At its core, RayTracer.jl is a general-purpose rendering engine, supporting both ray tracing and rasterization. Unlike prior differentiable rendering techniques, we maintain performance in the forward pass, ensuring usability for realistic applications.

Fig. 1: Utah Teapot Render from three different views. The camera definition shown in Listing can be easily modified to generate all these views.

RayTracer.jl provides a flexible API for specifying scenes, including lighting configuration, object geometry, materials, and camera positioning. Rendering proceeds from this scene definition, and outputs can be directly integrated with machine learning pipelines.

IV. Inverse Rendering

Inverse rendering entails estimating the parameters of a 3D scene from its rendered 2D image. With RayTracer.jl's differentiable pipeline, gradient-based optimization becomes feasible for various parameters, from camera pose to material colors.

Algorithm 1: Gradient-based optimization for inverse rendering

Input: Initial scene parameters, maximum iterations, loss tolerance, optimizer
Output: Optimized scene parameters

```
params ← Initial values
converged ← false; iter ← 0
while not converged and iter ¡ max_iter do
    loss ← mean_squared_loss(render_image(params), target_img)
    gs ← gradient(loss)
    for param in params do
        update!(optimizer, param, gs[param])
    end
    if loss ¡ tolerance then
        converged ← true
    end
    iter ← iter + 1
end
return params
```

V. Experiments

We now demonstrate several experiments utilizing RayTracer.jl's differentiable capabilities to solve inverse problems, including camera calibration, light estimation, and material recovery. All optimizations are performed using the Adam optimizer [9] and the Flux or Optim libraries [10].

A. Rendering Acceleration via BVH

To reduce computation during ray tracing, we adopt Bounding Volume Hierarchies (BVH) [11]. BVH allows pruning of the scene's search space when determining ray-object intersections.

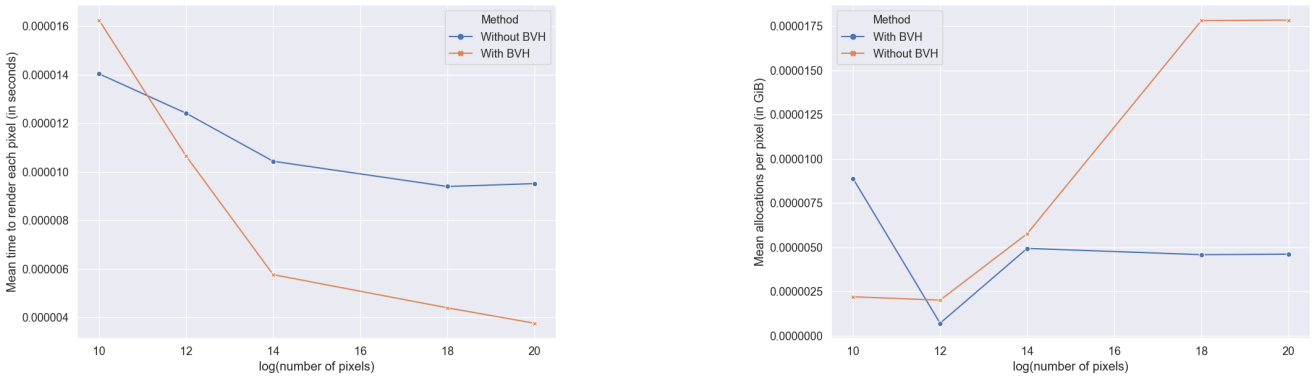
As shown in Fig. 2a and Fig. 2b, BVH provides significant performance and memory gains for higher-resolution images, with diminishing returns for small-scale scenes.

B. Automatic Differentiation vs Finite Differencing

This experiment benchmarks gradient computation via Zygote AD against Central Finite Differencing. We compute loss gradients w.r.t. scene parameters across increasing triangle counts. Fig. 3a and Fig. 3b illustrate that as parameters grow, AD vastly outperforms finite differencing, which becomes impractical beyond 50+ parameters.

C. Camera Parameter Calibration

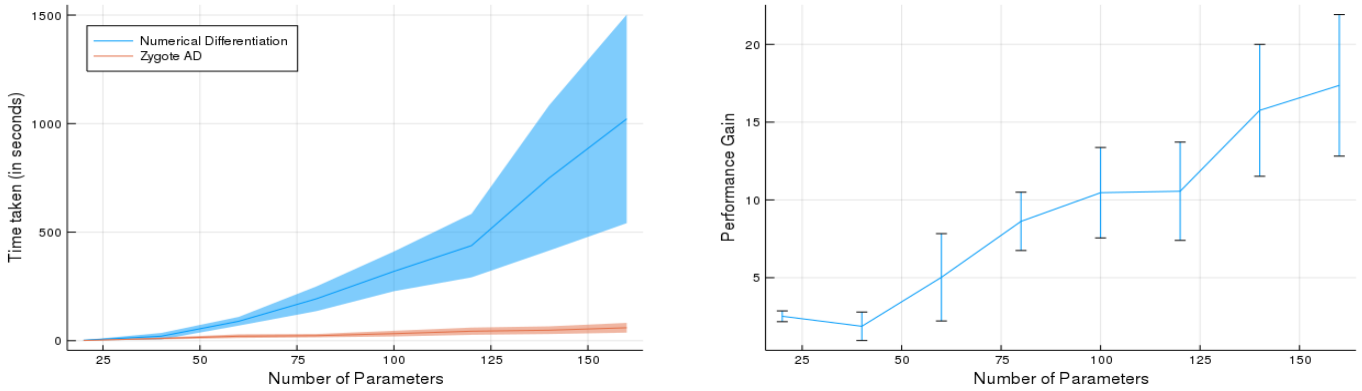
We optimize the camera position to match a given target image. The camera's initial guess and true configuration are provided in Listings, and Optimization reduces the loss from over 5700 to below 100, as seen in Fig. 1.



(a) Performance Benchmarks

(b) Memory Allocation Benchmarks

Fig. 2: Comparison between scenes rendered with and without BVH acceleration



(a) Backward Pass Time

(b) AD vs Numerical Differentiation

Fig. 3: Performance comparison: Zygote-based AD vs Finite Differences

D. Lighting Configuration Optimization

In this task, we recover the position and intensity of a point light given the scene and camera. The optimization process reconstructs the target image in under 500 steps.

E. Material Property Estimation

We optimize per-mesh diffuse color to match a reference image. Because color must remain in [0,1], we apply projected gradient descent after each update. shows rapid convergence—nearly matching the target after one iteration.

VI. Current Limitations

Despite our framework's flexibility, it struggles with inherently non-differentiable elements, such as mesh geometry reconstruction or complex global illumination effects. Additionally, optimizing across discrete topologies (e.g., triangle connectivity) remains an unsolved challenge. These limitations parallel those discussed in [3] and indicate areas for future research.

VII. Results

We evaluated *RayTracer.jl* across several differentiable rendering tasks to demonstrate its accuracy, performance, and integration capabilities with gradient-based optimization workflows. The experiments include benchmarking automatic differentiation against finite differences, and solving inverse problems involving camera calibration, lighting estimation, and material property recovery.

A. Performance Benchmarks

To evaluate forward rendering efficiency, we compared ray tracing performance with and without bounding volume hierarchy (BVH) acceleration. As illustrated in Fig. 2a, BVH significantly reduces rendering time for high-resolution scenes by minimizing the number of intersection tests. Additionally,

Fig. 2b shows a substantial reduction in memory usage due to efficient scene traversal. These results confirm the scalability of RayTracer.jl to complex scenes.

B. Differentiation Accuracy and Speed

The backward pass performance was assessed by comparing Zygote-based automatic differentiation with central finite differences. As shown in Fig. 3a, AD offers a substantial speed advantage, especially as the number of scene parameters increases. Fig. 3b highlights that Zygote provides lower approximation error and improved stability compared to finite differencing, which suffers from numerical instability and higher computational cost.

C. Camera Calibration

In the camera calibration task, the camera's position and orientation were optimized to match a known target image. The loss decreased from over 5700 to below 100 after several iterations, with

VIII. Conclusion

In this work, we introduced *RayTracer.jl*, a fully differentiable rendering engine developed in the Julia programming language. Our system capitalizes on Julia's strengths in high-performance numerical computing and seamless integration with modern automatic differentiation (AD) frameworks, particularly Zygote. This integration enables differentiability across the entire rendering pipeline without relying on manual derivation of gradients or approximations such as finite differencing.

By constructing the renderer from the ground up with differentiability as a core design principle, we achieve a general-purpose tool that not only supports realistic rendering through ray tracing and rasterization but also permits backpropagation of gradients with respect to arbitrary scene parameters. This makes it highly applicable in a wide range of computer vision and graphics tasks where inverse rendering is required.

We have demonstrated the capabilities of our framework through a series of controlled experiments, showcasing how gradients obtained via source-to-source AD can be effectively used to optimize lighting configurations, camera parameters, and material properties. Notably, our system achieves significant performance benefits when compared with traditional numerical differentiation methods, making it scalable to scenes with hundreds or thousands of differentiable parameters.

Furthermore, our renderer integrates naturally with deep learning libraries such as Flux, enabling its use in broader differentiable programming pipelines. This opens avenues for applications in neural rendering, self-supervised learning, differentiable simulation, and other domains where end-to-end differentiability is crucial.

Despite its strengths, RayTracer.jl inherits certain limitations common to differentiable renderers, including challenges with non-differentiable scene elements such as discrete geometry, occlusions, and global illumination effects. Nevertheless, the modular and extensible design of our framework allows for future expansion toward handling these complex cases.

To the best of our knowledge, RayTracer.jl is among the first renderers to employ source-to-source AD in Julia for full-scene differentiability. Our findings underscore the feasibility and efficiency of using high-level numerical languages like Julia to build performance-critical, gradient-aware tools that bridge graphics and machine learning. We believe this work lays a robust foundation for future research in inverse graphics, differentiable rendering, and the broader field of differentiable programming.

IX. Future Work

While *RayTracer.jl* provides a solid foundation for differentiable rendering using source-to-source automatic differentiation in Julia, several avenues remain open for further development and exploration.

First, one of the most prominent limitations of our current implementation is its inability to handle discontinuities and non-differentiable events robustly, such as visibility changes, hard shadows, and topological changes in geometry. Future work could involve integrating probabilistic or soft visibility techniques to enable more stable gradients in such scenarios, as explored in soft rasterization methods [12].

Second, our current renderer supports only basic forms of light transport such as direct illumination. Extending the system to incorporate global illumination, inter-reflections, and subsurface scattering would make it suitable for a broader class of photorealistic rendering tasks. However, doing so while maintaining differentiability and computational efficiency remains a challenging research problem.

Third, we aim to enhance interoperability with existing 3D asset pipelines by supporting more complex mesh formats, scene graphs, and physically-based material models. This would allow users to import more realistic scenes and conduct high-fidelity inverse rendering experiments.

In addition, improving the renderer's scalability to handle large-scale scenes with millions of primitives remains an important direction. This could involve optimizing memory layout, supporting GPU backends through libraries like CUDA.jl, and parallelizing computation-heavy routines such as ray–triangle intersection and BVH traversal.

Moreover, while our integration with Flux facilitates use in learning-based pipelines, more extensive experiments involving training neural networks with rendered gradients (e.g., for neural inverse rendering or 3D-aware generative models) would validate its utility in real-world machine learning applications.

Finally, we envision RayTracer.jl serving as a research platform for exploring hybrid rendering methods that combine rasterization and ray tracing under a unified differentiable framework. Bridging the gap between real-time and photorealistic rendering while preserving end-to-end differentiability would significantly enhance the versatility of the system.

REFERENCES

[1] A. Appel, "Some Techniques for Shading Machine Renderings of Solids," in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, ser. AFIPS '68 (Spring). New York, NY, USA: ACM, 1968, pp. 37–45. [Online]. Available: http://doi.acm.org/10.1145/1468075.1468082

[2] J. Pineda, "A Parallel Algorithm for Polygon Rasterization," in *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '88. New York, NY, USA: ACM, 1988, pp. 17–20. [Online]. Available: http://doi.acm.org/10.1145/54852.378457

[3] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen, "Differentiable Monte Carlo Ray Tracing Through Edge Sampling," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 222:1–222:11, Dec. 2018. [Online]. Available: http://doi.acm.org/10.1145/3272127.3275109

[4] M. Innes, "Don't Unroll Adjoint: Differentiating SSA-Form Programs," *CoRR*, vol. abs/1810.07951, 2018. [Online]. Available: http://arxiv.org/abs/1810.07951

[5] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A Fresh Approach to Numerical Computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017. [Online]. Available: https://doi.org/10.1137/141000671

[6] M. Johnson, R. Frostig, D. Maclaurin, and C. Leary, "JAX: Autograd and XLA," https://github.com/google/jax, 2018.

[7] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, Mar 2019. [Online]. Available: https://doi.org/10.1007/s12532-018-0139-4

[8] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah, "Fashionable Modelling with Flux," *CoRR*, vol. abs/1811.01457, 2018. [Online]. Available: http://arxiv.org/abs/1811.01457

[9] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations*, 12 2014.

[10] P. K Mogensen and A. N Riseth, "Optim: A mathematical optimization package for julia," *Journal of Open Source Software*, vol. 3, no. 24, p. 615, 4 2018. [Online]. Available: http://dx.doi.org/10.21105/joss.00615

[11] T. L. Kay and J. T. Kajiya, "Ray Tracing Complex Scenes," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 269–278. [Online]. Available: http://doi.acm.org/10.1145/15922.15916

[12] S. Liu, T. Li, W. Chen, and H. Li, "Soft rasterizer: A differentiable renderer for image-based 3d reasoning," *CoRR*, vol. abs/1904.01786, 2019. [Online]. Available: http://arxiv.org/abs/1904.01786