Caustic Projection Lamp - A Lamp That Is More Lamp

Evan Zhang
Cornell University
Ithaca, New York, USA
yz864@cornell.edu

Carrie Wang
Cornell University
Ithaca, New York, USA
jw2796@cornell.edu

Figure 1: Final Rendered Scene

1 INTRODUCTION

This project develops a caustic projection lamp that optimizes the shape of a cylindrical lens to project a user-defined image onto a surface when illuminated. Using differentiable rendering with Mitsuba 3 and Dr.Jit, the system performs inverse optimization to deform a mesh such that the resulting caustic pattern matches a target image. Instead of relying on handcrafted geometry or direct texture mapping, this approach formulates a physical light transport problem and solves it through gradient-based mesh displacement. The optimization includes a loss function comparing rendered output with the reference image, enabling precise, physically-based fabrication of custom projection optics.

2 PROBLEM

In prior work[mit [n. d.]], a flat rectangular glass panel placed outside a three-walled enclosure refracts a directional area light to project a specified image onto an interior wall. Although effective for generating caustics, this arrangement does not behave like a conventional lamp: the optimized refractor remains external to the room, its surface is strictly planar, and the illumination is supplied by a separate, off-axis source.

We propose to extend this concept by designing a cylindrical refractive element that houses its own light emitter. By embedding the source within a rotationally symmetric lens, the device can radiate light in all directions and produce immersive, spatially integrated caustic projections.

3 METHOD

3.1 Cylindrical Geometry

To initiate the setup, we first design a cylindrical lens geometry enclosed in a 2×2×2 bounding box. The lens is structured into two concentric layers to account for the double refraction that light undergoes before forming the target image projection. The inner layer is modeled manually in Blender and imported directly into the scene, serving as a fixed base that guides light entry. The outer

layer, which is subject to optimization, is programmatically generated using angular (U) and vertical (V) coordinates. The initial resolution is defined by a (u, v) pair and is further refined by a factor of four in each direction, resulting in a high-resolution mesh with fine-grained subdivisions. This layered design enables accurate control over light behavior and allows differentiable rendering techniques to iteratively adjust the outer lens shape while preserving a stable optical interface. In the later optimization process, the parameterized mesh is updated directly through vertex displacement informed by differentiable rendering, while the underlying topology and connectivity remain fixed throughout the process.

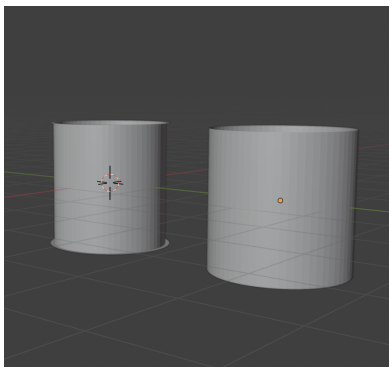

Figure 2: Geometry in Blender

3.2 Differentiable Rendering

This project employs differentiable rendering as the core method for optimizing the geometry of a custom projection lens. Differentiable rendering is a computational method that allows gradients to be propagated through the image formation process. Unlike traditional rendering—which treats the process of simulating light transport as a black box—differentiable rendering exposes the internal structure of rendering pipelines to automatic differentiation.

We implement this approach using the Mitsuba 3 renderer in combination with the Dr.Jit automatic differentiation framework. This differentiable rendering technique is used to simulate how light interacts with a cylindrical lens and forms a projection on nearby surfaces. As the scene is rendered, Mitsuba internally tracks the derivatives of pixel intensities with respect to the parameters controlling the lens surface.

3.3 Gradient-Based Optimization

The optimization of the lens geometry in this project is achieved using a gradient-based approach enabled by the Dr.Jit automatic differentiation framework. Once the rendered output is compared to the reference image, the system computes a loss representing their difference. Due to the differentiable rendering capabilities of Mitsuba 3, the vertex positions of the outer lens mesh can be computed automatically. These gradients are retrieved via dr.grad(verts) and used to update the vertex positions using a simple first-order gradient descent step. This process is embedded within an iterative optimization loop, which continues for a predefined number of steps.

4 EXPERIMENTS

4.1 Loss Design

We first define a joint absolute-error loss over the three projected views:

$$L_{\text{joint}} = \sum_{i=1}^{3} \sum_{(x,y)\in\Omega} \left| I^{(i)}_{\text{project}}(x,y) - I^{(i)}_{\text{gt}}(x,y) \right|.$$

We attempt to rebalance the contributions of each view by introducing non-negative weights w_i (with $\sum_i w_i = 1$):

$$L_w = \sum_{i=1}^{3} w_i \sum_{(x,y)\in\Omega} \left| I^{(i)}_{\text{project}}(x,y) - I^{(i)}_{\text{gt}}(x,y) \right|.$$

However, choosing appropriate w_i by hand is difficult:

- Different views can have vastly different error magnitudes, so fixed weights tend to either over-emphasize one view or under-represent another.
- Manual tuning of w_i for each scene is labor-intensive and does not generalize across lighting conditions or geometries.
- A poor weight choice can bias the optimization toward the highest-intensity view, degrading performance on others.

For these reasons, we avoid a weighted sum and instead use a cyclic per-view schedule that requires no manual parameters.
Next, to remove sensitivity to global intensity scale, we normalize each view before computing squared-error. Define for each view i:

$$\tilde{I}^{(i)} = \frac{I^{(i)}_{\text{project}}}{\mu\big(I^{(i)}_{\text{project}}\big)}, \quad \tilde{R}^{(i)} = \frac{I^{(i)}_{\text{gt}}}{\mu\big(I^{(i)}_{\text{gt}}\big)},$$

where $\mu(\cdot)$ is the mean over all $N = |\Omega|$ pixels. The scale-independent per-view loss is

$$L^{(i)}_{\text{si}} = \frac{1}{N} \sum_{(x,y)\in\Omega} \big(\tilde{I}^{(i)}(x,y) - \tilde{R}^{(i)}(x,y) \big)^2.$$

Finally, at iteration t we select exactly one view to back-propagate, cycling through $i = 0, 1, 2$:

$$i_t = t \bmod 3, \qquad L^{(t)} = L^{(i_t)}_{\text{si}}.$$

This cyclic strategy ensures each view receives equal optimization effort, without any hand-tuned weights.

4.2 Scene Design

We evaluate our method in a synthetic "caustic room" environment, consisting of a central lens and three planar targets (front wall, left wall, right wall, plus the floor). The scene design and caustics experiment progress through four stages:

(1) **Testing lens (un-optimized) light penetration.**
To evaluate transmission through our new cylindrical geometry, we replaced the original flat optic with the meshgrid-wrapped cylinder and placed a white point light at its center. In this first test, the illumination exhibited pronounced dark bands. We determined that these stripes align precisely with the mesh's subdivision boundaries.

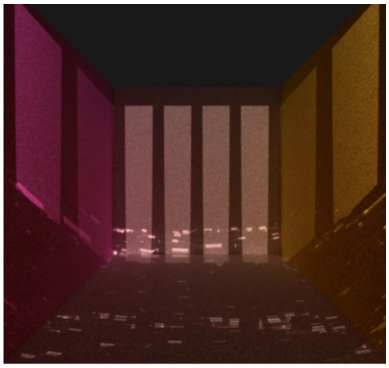

Figure 3: Un-optimized lens light penetration.

(2) **Testing radial lighting in the center of the room.** Since we were worried that the light source and the lens would be too close to the walls if the lamp is moved to the center of the room, we want to test how large the projections are before training. From this experiment, we can pick the sensor's field-of-view and mounting distance that won't cut anything off.

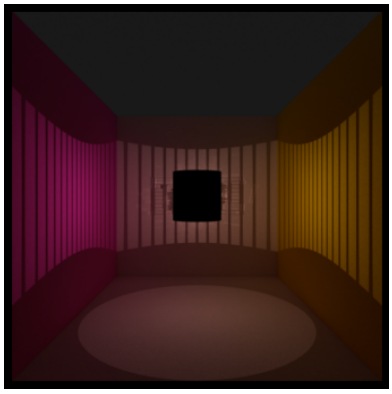

Figure 4: Radial lighting pattern at the room center.

(3) **Testing back wall image projection.** Next, we projected a real target image onto the back wall. To our surprise, as soon as optimization kicked in the heightmap started "patching up" those dark stripes—literally pushing the surface geometry to steer light into the gaps and smooth out the bands.

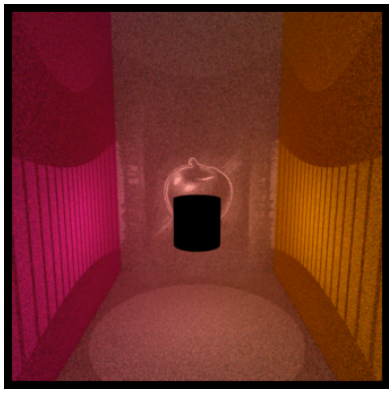

Figure 5: Back wall projection.

(4) **Testing projection on all three walls.** Finally, we added two extra sensors pointing at the left and right walls so the optimizer can "see" those surfaces too.

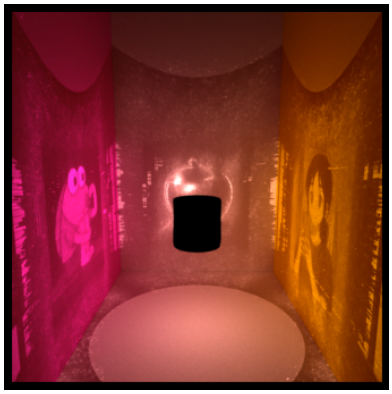

Figure 6: Final projections on front, left, and right walls.

5 RESULT

5.1 Displacement "“patching up" dark stripes

During optimization, the heightmap locally steepens at mesh boundaries to redirect light into the dark bands, automatically eliminating the stripes. This correction is because the optimization tries to minimize the scale-independent loss, with no manual tuning required.

Before the optimization starts, the rendered image is just noise with large black bands.

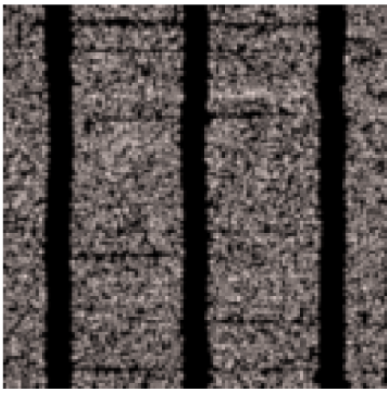

Figure 7: 0 Iterations

The optimizer begins to “patch” the stripes and reduce noise, and a faint apple silhouette emerges along the central bands.

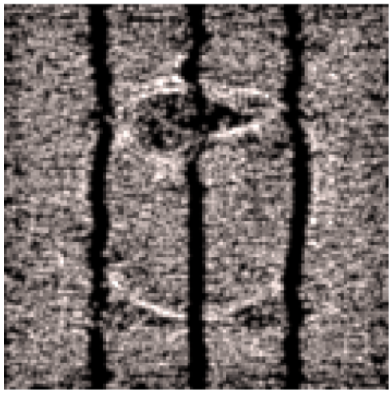

Figure 8: 2000 Iterations

All dark bands are filled and the apple caustic projection becomes sharp and high-contrast, indicating convergence to the target pattern.

Figure 9: 5000 Iterations

5.2 Displaced lens

Figure below shows the "bumpy" optimized cylindrical lens geometry. Because the heightmap is defined over the cylinder’s UV parameterization, each UV coordinate corresponds to a unique outgoing ray direction that Mitsuba’s differentiable renderer routes to one of the three sensors (front, left, or right). During optimization, per-pixel losses computed by each sensor back-propagate only to the heightmap samples at those UV coordinates, so the central band deforms to project onto the back wall and the two side bands deform to project onto the left and right walls, respectively.

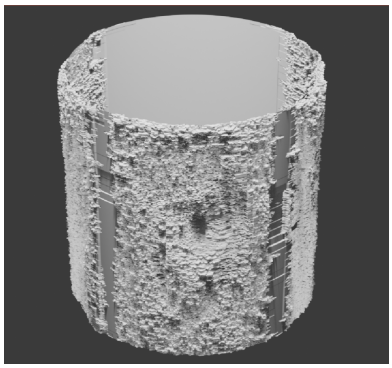

Figure 10: Displaced Lens

6 CONCLUSION

This project presents a method for optimizing cylindrical lens geometries using differentiable rendering. Unlike traditional planar setups, our cylindrical design enables 360-degree caustic projection through gradient-based deformation of a parameterized outer mesh. The use of Mitsuba 3 and Dr.Jit allows gradients to flow through the entire rendering pipeline, guiding vertex-level displacements that result in multi-view projections without manual tuning. This approach opens new possibilities for fabricating spatially immersive lighting devices that are both visually expressive and physically grounded.

REFERENCES

[n. d.]. Mitsuba 3 Documentation: Caustics Optimization. https://mitsuba.readthedocs.io/en/stable/src/inverse_rendering/caustics_optimization.html. Accessed: 2025-05-15.