Differentiable Rendering using RGBXY Derivatives and Optimal Transport

JIANKAI XING, BNRist, Department of CS&T, Tsinghua University, China
FUJUN LUAN, Adobe Research, USA
LING-QI YAN, University of California, Santa Barbara, USA
XUEJUN HU, BNRist, Department of CS&T, Tsinghua University, China
HOUDE QIAN, BNRist, Department of CS&T, Tsinghua University, China
KUN XU*, BNRist, Department of CS&T, Tsinghua University, China

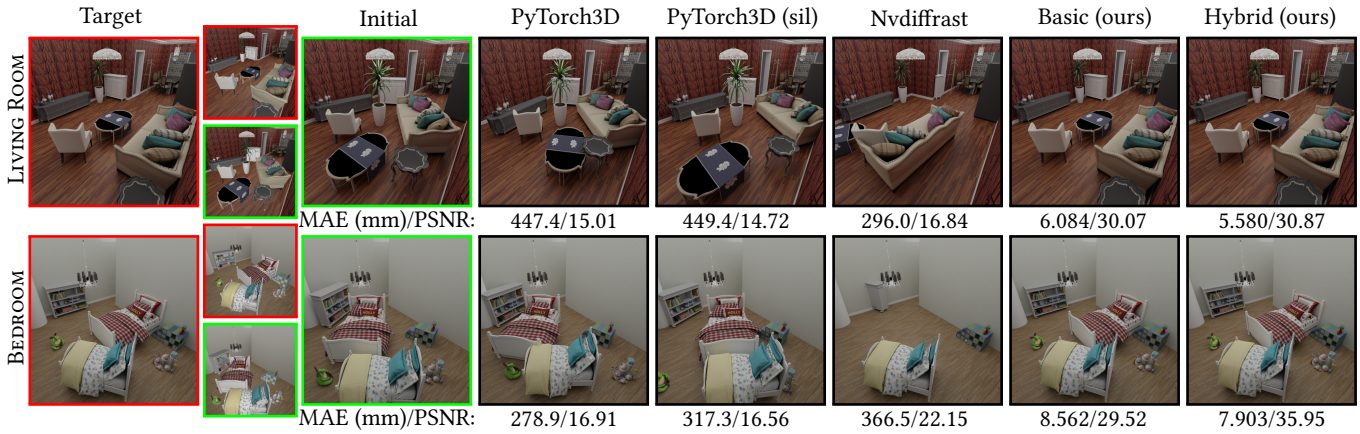


Fig. 1. Automatic furniture layout adjustment via differentiable rendering. Given a target image with the desired furniture layout, we initialize the scene with randomly placed furniture models in the room and optimize for their transformations (i.e., translation within X-Z plane and rotation around Y axis) using existing differentiable renderers with standard color derivatives vs. our method using RGBXY derivatives. PyTorch3D [Ravi et al. 2020] fails to match the target even with silhouette masks provided. Nvdiffrast [Laine et al. 2020] produces slightly better results with the gradients from analytic antialiasing, but often gets stuck in local minima with suboptimal convergence. In contrast, our method consistently outperforms existing methods thanks to the RGBXY derivatives, yielding accurately matched results to the reference. Lastly, we show that our method can be combined with traditional methods through a hybrid optimization strategy, further improving accuracy among a variety of inverse rendering tasks. (Note that optimization is performed using rasterized images with a resolution of 256×256, as shown in the second column, while other images are re-rendered in a high resolution with ray tracing for higher display quality).

Traditional differentiable rendering approaches are usually hard to converge in inverse rendering optimizations, especially when initial and target object locations are not so close. Inspired by Lagrangian fluid simulation, we present a novel differentiable rendering method to address this problem. We associate each screen-space pixel with the visible 3D geometric point covered by the center of the pixel and compute derivatives on geometric points rather than on pixels. We refer to the associated geometric points as point proxies of pixels. For each point proxy, we compute its 5D RGBXY derivatives which measures how its 3D RGB color and 2D projected screen-space position change with respect to scene parameters. Furthermore, in order to capture global and long-range object motions, we utilize optimal transport based pixel matching to design a more sophisticated loss function. We have conducted experiments to evaluate the effectiveness of our proposed method on various inverse rendering applications and have demonstrated superior convergence behavior compared to state-of-the-art baselines.

CCS Concepts: • **Computing methodologies** → **Rendering**.

Additional Key Words and Phrases: differentiable rendering, RGBXY derivatives, optimal transport

ACM Reference Format:
Jiankai Xing, Fujun Luan, Ling-Qi Yan, Xuejun Hu, Houde Qian, and Kun Xu. 2022. Differentiable Rendering using RGBXY Derivatives and Optimal Transport. *ACM Trans. Graph.* 41, 6, Article 189 (December 2022), 13 pages. https://doi.org/10.1145/3550454.3555479

*Kun Xu is the corresponding author.

Authors' addresses: Jiankai Xing, xjk21@mails.tsinghua.edu.cn, BNRist, Department of CS&T, Tsinghua University, Beijing Shi, China; Fujun Luan, luanfj11@gmail.com, Adobe Research, USA; Ling-Qi Yan, lingqi@cs.ucsb.edu, University of California, Santa Barbara, USA; Xuejun Hu, huxj19@mails.tsinghua.edu.cn, BNRist, Department of CS&T, Tsinghua University, Beijing Shi, China; Houde Qian, qhd19@mails.tsinghua.edu.cn, BNRist, Department of CS&T, Tsinghua University, Beijing Shi, China; Kun Xu, xukun@tsinghua.edu.cn, BNRist, Department of CS&T, Tsinghua University, Beijing Shi, China.

0730-0301/2022/12-ART189
https://doi.org/10.1145/3550454.3555479

1 INTRODUCTION

Recent advances in differentiable rendering bridge the gap between photorealistic image synthesis of 3D models and inverse rendering of these model parameters through iterative gradient descent. Modern computer graphics pipelines can synthesize images either by ray tracing the 3D scene for light transport simulation (e.g., VFX applications) or rasterizing the renderable primitives to map the scene geometry to pixels (e.g., real-time game engines). These forward rendering pipelines can be made *differentiable* with respect to the 3D scene parameters, including camera pose, geometry, material appearance and illumination. As a result, one can optimize and recover such unknown scene parameters through gradient-based optimization on some predefined image loss functions between the rendered image and target reference (e.g., captured photos), where gradients are computed using a differentiable renderer.

However, despite that differentiable renderers have demonstrated a broad impact on various inverse rendering applications such as object capture [Munkberg et al. 2021] and 3D model simplification [Hasselgren et al. 2021], we would like to note that existing differentiable rendering approaches typically rely on per-pixel image loss functions (such as L2 loss) from a pair of RGB images and compute per-pixel color derivatives with respect to desired scene parameters. Such per-pixel color derivatives are intrinsically sparse and local, leading to less robustness in optimization — it is naturally well suited for fine-scale geometry refinement (e.g., displacement) with sufficiently overlapped and aligned image regions, while quickly becoming less effective for optimizing global, long-range object translation and rotation when the rendered image and the target reference are far from a perfect per-pixel alignment. For instance, given a scene containing a sphere, let us set the goal to be optimizing for a translation offset that moves the sphere from left (initial state) to right (target state), when our initialization of the sphere location is not so close to the target, the rendered sphere will have no overlapped image pixels compared to the reference sphere, yielding an unchanged L2 loss with zero gradients that break the optimization.

To address this problem, we draw inspiration from Lagrangian fluid simulation. Our key idea is to associate each pixel with a *point proxy* and compute derivatives on point proxies instead of on a fixed grid of pixels. A point proxy of a pixel refers to the visible 3D geometric point on the mesh covered by the center of the pixel. First, for each point proxy, we derive its 5D RGBXY derivatives to measure how its 3D RGB color and 2D XY screen-space position change with respect to desired scene parameters. Second, we leverage optimal transport to find a pixel matching between rendered and target images and design a novel loss function based on the pixel matching. Our RGBXY derivatives are dense and could account for global and long-range object motions through the optimal transport based loss function, naturally leading to better robustness in optimization. Taking the sphere scene mentioned above as an example, on one hand, optimal transport helps find the correspondences between the rendered sphere and the reference sphere, on the other hand, by including screen-space positions (XY) in the definition of RGBXY derivatives, we could explicitly drive the sphere moving from left to right, even when their initial and target positions are far away.

To validate our theory and algorithms, we compare them with state-of-the-art approaches on a set of applications and demonstrate that our method consistently improves the result. Our contributions include:

- we derive novel 5D RGBXY derivatives on point proxies;
- we propose an optimal transport based loss function that considers global and long-range pixel correspondences;
- based on the 5D RGBXY derivatives and the optimal transport based loss function, we present a novel differentiable rendering method which has demonstrated to have more robust convergence behavior compared to state-of-the-art works and can be easily integrated into existing differentiable rendering frameworks.

The code of our method is available at https://www.github.com/jkxing/DROT.

2 RELATED WORK

2.1 Differentiable Rendering

Inverse rendering requires formulating a novel forward parametric model and computing derivatives with respect to its parameters, specifically for each reconstruction problem via analysis-by-synthesis techniques. Recently, there is a surging interest in developing fully-differentiable forward rendering methods in graphics and vision community, also known as *differentiable rendering*, to facilitate statistical inference and deep learning pipelines (e.g., the decoder network of an auto-encoder architecture [Che et al. 2020]) in an end-to-end fashion, enabling a wide range of practical inverse rendering applications such as object capture [Cai et al. 2022; Deng et al. 2022; Luan et al. 2021; Munkberg et al. 2021] and material estimation [Gao et al. 2019; Guo et al. 2020; Shi et al. 2020]. Please refer to Kato et al. [2020a] and Zhao et al. [2021] for a more comprehensive survey.

The first general-purpose differentiable rendering frameworks such as OpenDR [Loper and Black 2014] and Neural 3D Mesh renderer [Kato et al. 2018] have been developed to enable analytical differentiation at the cost of utilizing *approximate* forward models, which focus on primary visibility and cannot handle complex materials or light transport effects. On the other hand, one technical difficulty in differentiating Monte Carlo renderers (e.g., a path tracer) is proper handling of edge derivatives, typically due to geometry discontinuities around the object silhouettes and depth boundaries, where simple automatic differentiation can only handle interior gradients while completely missing the boundary term [Zhang et al. 2020], yielding inaccurate gradient estimates and suboptimal convergence consequently.

Due to the aforementioned challenge, there have been mainly two types of differentiable rendering methods that are developed to tackle this problem, namely *physics-based differentiable rendering* and *differentiable rasterization*.

Physics-based differentiable rendering. This type of method aims at fully differentiating a path tracer that handles global illumination through light transport simulation. Li et al. [2018] propose the edge sampling algorithm that estimates the boundary gradient term using Monte Carlo sampling. This approach has been further reformulated

in path space using the Reynolds transport theorem [Zhang et al. 2020, 2021]. Alternatively, fast approximation of boundary term by sampling the continuous area instead of the silhouette can be achieved through reparameterized integrals [Bangaru et al. 2020; Loubet et al. 2019]. Vicini et al. [2022] further design and apply efficient reparameterization for SDF representations that enable accurate shape parameter derivatives. These approaches can handle complex light transport effects such as caustics and glossy interreflection, while typically being computationally expensive and heavy in implementation.

Differentiable rasterization. This type of method rasterizes renderable primitives in a scene by projecting and shading mesh vertices on the image plane with automatic differentiation. To handle the geometry discontinuities around mesh silhouettes, triangles along camera rays are each assigned a transparency value (a.k.a. "softening") based on a signed distance function and blended through a differentiable aggregating process, such as SoftRas [Liu et al. 2019] and PyTorch3D [Ravi et al. 2020], enabling efficient gradient backpropagation to multiple triangles along a ray. Nvdiffrast [Laine et al. 2020] uses analytic antialiasing method on top of rasterization to generate reliable visibility gradients. It achieves much higher performance than previous frameworks through a highly optimized hardware implementation. Existing differentiable rasterization methods could be combined with preconditioned gradient descent for stably optimizing the vertex positions of meshes [Nicolet et al. 2021]. In this work, we mainly use differentiable rasterizers as the backend of our differentiable rendering method, for the ease of implementation and fast speed in gradient-based optimization.

Note that, current differentiable rendering approaches utilize per-pixel image derivatives which are computed locally pixels on a set of fixed grid locations. Therefore, such per-pixel derivatives cannot help with global and long-range optimization during inverse rendering when few or no pixels are overlapped between the image pairs.

2.2 Optimal Transport

Optimal transport [Kantorovich 1942; Monge 1781], in its discrete form, gives a framework to measure the "distance" between two distributions, where each distribution is discretized into a set of points, each carrying an amount of "mass". In order to minimize the distance, optimal transport solves for the amount of mass transported between pairs of points from the two distributions, respectively.

Recent works in computer graphics have applied optimal transport to various subareas [Bonneel et al. 2016, 2011; Solomon et al. 2015]. Since optimal transport often computes slowly, approximate solutions have been proposed to enable faster calculation, such as Sinkhorn divergences [Cuturi 2013], convolutional Wasserstein distances [Solomon et al. 2015], Sliced Optimal Transport (SOT) [Paulin et al. 2020] and Sliced Partial Optimal Transport (SPOT) [Bonneel and Coeurjolly 2019].

We are especially interested in a special case of optimal transport that finds one-to-one mapping. In this case, the optimal transport is equivalent to finding a minimum weight matching in a bipartite graph, but can be performed in a much more efficient way with a slight approximation using Sinkhorn divergences [Feydy et al. 2019]. Since optimal transport provides global optimal mapping, which is automatically shape-aware, the acquired mapping provides us with nice correspondence for our method.

It is worth mentioning that the goal of optimal transport is also closely related to bipartite graph matching [Crouse 2016; Kuhn and Yaw 1955] and optical flow [Ilg et al. 2017; Truong et al. 2022]. In our experiments, we have demonstrated the superiority of optimal transport over alternative techniques for our task.

2.3 Fluid Simulation

Since our idea is inspired by fluid simulation, we provide a brief discussion of it. There are two viewpoints towards the motion of a continuous fluid [Bridson 2008], including the Eulerian viewpoint and the Lagrangian viewpoint. The Eulerian viewpoint, i.e., grid-based fluid simulation methods [Fedkiw et al. 2001; Stam 1999], focuses on fixed locations in space and tracks how fluid quantities (i.e., densities, velocities, temperatures, etc.) change with time at these fixed locations. Differently, the Lagrangian viewpoint, i.e., smoothed particle hydrodynamics (SPH) based methods [Desbrun and Gascuel 1996; Müller et al. 2003; Premžoe et al. 2003], considers the fluid as a particle system, where each particle could be thought as a molecule of the fluid. The Lagrangian viewpoint tracks how the position, velocity and other associated quantities of each particle change with time.

The Eulerian viewpoint and the Lagrangian viewpoint could be connected through the *material derivative* [Bridson 2008]:

$$\frac{Ds}{Dt} = \frac{\partial s}{\partial t} + \nabla s \cdot \frac{d\mathbf{p}}{dt}, \tag{1}$$

where t denotes the time, $\mathbf{p}$ and s denote the position and a specific quantity (e.g., temperature) of a particle, respectively, and $\nabla(\cdot)$ denotes the spatial gradient operator. The *Eulerian derivative* $\partial s/\partial t$ tracks how the quantity at the fixed location changes with time. The material derivative Ds/Dt tracks how the quantity of the particle changes with time, and $d\mathbf{p}/dt$ is the positional derivative (or velocity) of the particle.

We draw inspiration from Lagrangian fluid simulation and define derivatives on movable geometric points rather than on pixels at fixed locations.

3 BACKGROUND AND MOTIVATION

In this section, we first briefly review existing differentiable renderers in Sec. 3.1, then we present the motivation of our method in Sec. 3.2.

In the following, we focus on differentiable rasterizers. Support for ray tracing or other geometry types such as implicit surfaces remains future work.

3.1 Background

3.1.1 Formulation. Given a 3D scene containing mesh-based geometries, lights, materials, textures, cameras, etc, a rasterizer renders a 2D image $\mathbf{I}$ depicting the scene. The final RGB color $\mathbf{c} = (r, g, b)$ of each screen-space pixel $\mathbf{p} = (x, y)$ of the rendered image could be formulated by:

$$\mathbf{I}(\mathbf{p}) = \mathbf{c} = f(\mathbf{p}, \Theta), \tag{2}$$

where $f(\cdot)$ denotes the complex rendering function that includes computations of shading, interpolation, projection, filtering (antialiasing), etc., and Θ denotes the set of scene parameters such as camera pose, light direction, vertex position, texture color, etc. For simplicity, we interchangeably denote the color of a pixel $\mathbf{p}$ using $\mathbf{I}(\mathbf{p})$ or using symbol $\mathbf{c}$.

A differentiable renderer augments the rasterizer by additionally providing derivatives with respect to specific scene parameters besides rendered pixel colors. Specifically, it provides the color derivative $\partial\mathbf{I}(\mathbf{p})/\partial\theta$ (short as $\partial\mathbf{c}/\partial\theta$) at each pixel $\mathbf{p}$ where θ is a specific scene parameter. Note that the derivative $\partial\mathbf{c}/\partial\theta$ is computed at fixed screen-space locations, and we refer to it as the *standard color derivative*, short as *standard derivative*.

The standard color derivatives for the whole image $\partial\mathbf{I}/\partial\theta$ could also be given by:

$$\frac{\partial \mathbf{I}}{\partial \theta} = \left[\frac{\partial \mathbf{I}(\mathbf{p_1})}{\partial \theta}, \cdots, \frac{\partial \mathbf{I}(\mathbf{p_N})}{\partial \theta}\right] = \left[\frac{\partial \mathbf{c_1}}{\partial \theta}, \cdots, \frac{\partial \mathbf{c_N}}{\partial \theta}\right], \tag{3}$$

where N is the number of pixels, $\mathbf{p_i}$ denotes the i-th pixel and $\mathbf{c_i}$ denotes the color of pixel $\mathbf{p_i}$.

3.1.2 Loss function. In a typical inverse rendering problem, the goal is to recover some specific scene parameters through gradient based optimization on a predefined scalar loss function. The loss function L is usually defined as the sum of pixel-wise differences between the rendered image $\mathbf{I}$ and a given target reference image $\mathbf{I^{ref}}$, such as an L2 loss:

$$L(\mathbf{I}) = \sum_{i=1}^{N} L(\mathbf{p_i}), \quad \text{where} \quad L(\mathbf{p_i}) = (\mathbf{I}(\mathbf{p_i}) - \mathbf{I^{ref}}(\mathbf{p_i}))^2. \tag{4}$$

The differentiable renderer could then compute the derivative of loss function L with respect to a scene parameter θ through the chain rule:

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \mathbf{I}} \cdot \frac{\partial \mathbf{I}}{\partial \theta}. \tag{5}$$

The final scene parameters could be recovered by optimizing the loss function L through an iterative gradient descent process with the above derivative $\partial L/\partial\theta$.

3.2 Motivation

While traditional differentiable renderers have demonstrated very nice performance on many inverse rendering tasks, they quickly become less robust for optimizing global, long-range object transformation. Recall the simple example of a scene containing a single sphere: if the initial sphere location is far away from the target location, i.e., there are no overlapped pixels between the initial sphere and target sphere, the L2 loss would simply produce zero gradients and the optimization would probably get stuck in undesired local minima. The reason for such problem is due to the fact that pixels are at fixed screen space locations and the standard derivatives computed on pixels are intrinsically local. While the problem could be alleviated through blurring with soft boundaries [Liu et al. 2019] or multi-resolution strategies [Li et al. 2018], it is still not robust enough.

This motivated us to seek for a more robust differentiable rendering method that could deal with global and long-range object motions.

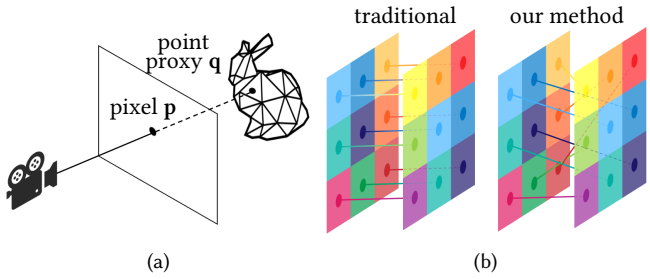


Fig. 2. Illustrations. (a) we associate each pixel with (at most) one geometric point — the underlying visible 3D geometric point on the mesh. (b) we employ optimal transport to find a pixel matching between rendered and target images while traditional methods could be viewed as always using an identity matching.

4 METHOD

In this section, we first present the overview of our method in Sec. 4.1, after that, we introduce the 5D RGBXY derivatives in Sec. 4.2 and the optimal transport based loss function in Sec. 4.3.

4.1 Overview of our method

The problem setup and pipeline of our method are almost the same as traditional differentiable rasterizers (Sec. 3.1). Considering an inverse rendering task, our goal is to optimize for some specific scene parameters through iterative gradient descent, so that the rendered image could be as close as possible to a given target reference image. The major difference lies in that our method uses movable geometric points instead of fixed-location pixels as the primitives for computing derivatives and loss functions.

Specifically, we associate each pixel with (at most) one geometric point — the visible 3D geometric point on the mesh covered by the center of the pixel, namely *point proxy*, as shown in Fig. 2 (a). For each point proxy, we derive its 5D RGBXY derivatives, which measures how its color (RGB) and its projected screen-space position (XY) change with respect to desired scene parameters. The RGBXY derivatives could track not only its appearance change but also its movement. See Sec. 4.2 for details.

Furthermore, in order to explicitly drive each point proxy to the correct location, we employ optimal transport to obtain a pixel matching between a pair of rendered images and target reference images. Then, we design a novel loss function by considering both color and positional differences between matched pixels. See Sec. 4.3 for details.

A simple illustration of the difference between our method and traditional differentiable rasterizers is given in Fig. 2 (b). Traditional methods could be viewed as always using an identity matching between rendered and target images, in contrast, we use optimal transport based pixel matching to define a more sophisticated loss function.

4.2 RGBXY Derivatives

As described in Sec. 4.1, instead of computing derivatives at fixed screen space locations (i.e., pixels), we associate each pixel with a point proxy (i.e., the visible underlying 3D geometric point on the

mesh it covers) and compute the RGBXY derivatives on the point proxy with respect to scene parameters.

Specifically, considering a point proxy $\mathbf{q}$ on the mesh, denoting its projected screen-space pixel position as $\mathbf{p} = \mathbf{p}(\mathbf{q}) = (x, y)$, and its observed RGB color as $\mathbf{c} = \mathbf{c}(\mathbf{q}) = (r, g, b)$, we define its 5D RGBXY derivative as the combination of a 3D color derivative and a 2D positional derivative:

$$\left[\frac{D\mathbf{c}}{D\theta}, \frac{\partial \mathbf{p}}{\partial \theta}\right], \tag{6}$$

where $D\mathbf{c}/D\theta$ and $\partial\mathbf{p}/\partial\theta$ measure how the RGB color and the projected screen-space position of the point proxy change with respect to scene parameters θ, respectively. Imagine that when scene parameters change, the point proxy always corresponds to the same underlying 3D geometric point on the mesh, i.e., the 3D point on the same triangle with the same interpolating barycentric weights. We refer to $D\mathbf{c}/D\theta$ as the *material derivative*, and $\partial\mathbf{p}/\partial\theta$ as the *positional derivative*, respectively.

4.2.1 Computation of RGBXY derivatives. Denoting the three vertices of the triangle which $\mathbf{q}$ belongs to as $\mathbf{v_i}$ $(1 \le i \le 3)$, it is easy to know that:

$$\mathbf{q} = w_1\mathbf{v_1} + w_2\mathbf{v_2} + w_3\mathbf{v_3}, \tag{7}$$

where w_i $(1 \le i \le 3)$ are barycentric weights satisfying $\sum_i w_i = 1$. Note that the barycentric weights of $\mathbf{q}$ are considered as unchanged by definition, so that we have $\partial w_i/\partial\theta = 0$. Accordingly, the positional derivative could be computed by:

$$\frac{\partial \mathbf{p}}{\partial \theta} = \left[\frac{\partial h(\mathbf{q})}{\partial \mathbf{q}}\right] \cdot \left[\frac{\partial \mathbf{q}}{\partial \theta}\right] = \left[\frac{\partial h(\mathbf{q})}{\partial \mathbf{q}}\right] \cdot \left[w_1\frac{\partial \mathbf{v_1}}{\partial \theta} + w_2\frac{\partial \mathbf{v_2}}{\partial \theta} + w_3\frac{\partial \mathbf{v_3}}{\partial \theta}\right], \tag{8}$$

where $h(\cdot)$ is the projection function that maps the 3D point $\mathbf{q}$ to screen space position $\mathbf{p}$, i.e., $\mathbf{p} = h(\mathbf{q})$.

Similarly, the observed color $\mathbf{c}$ at point proxy $\mathbf{q}$ could be generally expressed as a shading model as below:

$$\mathbf{c} = g(\mathbf{a}) = g(w_1\mathbf{a_1} + w_2\mathbf{a_2} + w_3\mathbf{a_3}), \tag{9}$$

where $\mathbf{a}, \mathbf{a_i}$ denote specific attributes at $\mathbf{q}$ and at the three triangle vertices $\mathbf{v_i}$ $(1 \le i \le 3)$, respectively. The attributes may include normal, albedo and other shading parameters and $g(\cdot)$ denotes a specific shading model. The attributes $\mathbf{a}$ are computed through barycentric interpolation from the attributes at the three triangle vertices.

Accordingly, we could compute the material derivative through:

$$\frac{D\mathbf{c}}{D\theta} = \left[\frac{\partial g}{\partial \mathbf{a}}\right] \cdot \left[\frac{D\mathbf{a}}{D\theta}\right] = \left[\frac{\partial g}{\partial \mathbf{a}}\right] \cdot \left[w_1\frac{\partial \mathbf{a_1}}{\partial \theta} + w_2\frac{\partial \mathbf{a_2}}{\partial \theta} + w_3\frac{\partial \mathbf{a_3}}{\partial \theta}\right]. \tag{10}$$

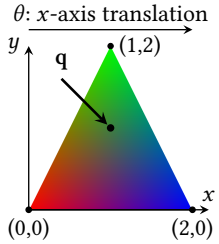
Similar to computing the positional derivative in Eq. 8, the barycentric weights are kept unchanged and their derivatives are naturally zero (i.e. $\partial w_i/\partial\theta = 0$) which will not contribute to the material derivative.

4.2.2 Relationship between RGBXY derivatives and standard derivatives $\partial\mathbf{c}/\partial\theta$. There are two major differences between our RGBXY derivatives and standard color derivatives. First, RGBXY derivatives are defined on geometric points associated with pixels while standard derivatives are defined on pixels. Second, RGBXY derivatives are 5D, measuring the changes of both colors and positions with respect to scene parameters. In contrast, standard derivatives track only color changes observed at fixed screen-space locations.

Despite the differences, the standard derivatives on a pixel could be connected to the RGBXY derivatives on the point proxy of that pixel, through:

$$\frac{\partial \mathbf{c}}{\partial \theta} = \frac{D\mathbf{c}}{D\theta} - \frac{\partial \mathbf{c}}{\partial \mathbf{p}} \cdot \frac{\partial \mathbf{p}}{\partial \theta} = \frac{D\mathbf{c}}{D\theta} - \nabla\mathbf{c} \cdot \frac{\partial \mathbf{p}}{\partial \theta}, \tag{11}$$

where $\nabla\mathbf{c}$ denotes the spatial gradient of color, i.e., how image color changes with screen space locations. Notice that the form of the above equation is rather similar to Eq. 1 in fluid simulation.



Let us look at a simple example (shown on the left) to better understand the concepts of all types of derivatives and their relationships. There is a colored triangle in the screen space with only one scene parameter θ which controls the amount of translation in the x-direction. Its three vertices are located at $(0, 0)$, $(2, 0)$, and $(1, 2)$, respectively. Their colors are pure red, blue, and green, respectively. Let's consider the center point of the triangle $\mathbf{q}$ which is projected at pixel $\mathbf{p}$ in screen space. Considering the standard derivative on pixel $\mathbf{p}$ and our RGBXY derivatives on its point proxy $\mathbf{q}$, it is easy to obtain that the standard derivative $\partial\mathbf{c}/\partial\theta = (0.5, 0, -0.5)$, our material derivative $D\mathbf{c}/D\theta = \mathbf{0}$, and our positional derivative $\partial\mathbf{p}/\partial\theta = (1, 0)$, satisfying Eq. 11.

In this example, we could observe a seemingly conflicting fact that the material derivative $D\mathbf{c}/D\theta$ is zero but the standard derivative $\partial\mathbf{c}/\partial\theta$ is non-zero. The zero-valued material derivative is simply due to the color of the underlying geometry point is always unchanged when the triangle moves. However, the color observed at a fixed pixel location could be changed due to the movement of the triangle, leading to a non-zero standard derivative.

4.3 Optimal Transport based Loss Function

In order to capture global and long-range correspondences, instead of using pixel-wise loss functions such as L1/L2 losses (Eq. 4), we propose to use a more sophisticated loss function based on optimal transport.

4.3.1 Our loss function. Discrete optimal transport algorithms solve the following transportation problem. Imagine there are N suppliers where the amount of "mass" of supplier i is x_i $(1 \le i \le N)$, and there are M consumers where the amount of demanding mass of consumer j is y_j $(1 \le j \le M)$, satisfying $\sum_{i=1}^{N} x_i = \sum_{j=1}^{M} y_j$. The goal is to find a transportation matrix T with minimal overall transportation cost L:

$$L = \sum_{i,j=1}^{N,M} T_{ij}c(i, j), \quad \text{subject to} \quad \sum_{j=1}^{M} T_{ij} = x_i, \sum_{i=1}^{N} T_{ij} = y_j, \tag{12}$$

where T_{ij} and $c(i, j)$ denote the transportation amount, and unit transportation cost from supplier i to consumer j, respectively.

Our scenario naturally fits to the formulation of the optimal transport problem. Consider all the pixels in the rendered image $\mathbf{I}$

as suppliers and all the pixels in the target image $\mathbf{I}^{\mathbf{ref}}$ as consumers, we simply set all $x_i = 1$ and all $y_j = 1$ (we assume $\mathbf{I}$ and $\mathbf{I}^{\mathbf{ref}}$ have the same number of pixels so that $N = M$). The unit transportation cost from pixel $\mathbf{p_i}$ in the rendered image $\mathbf{I}$ to pixel $\mathbf{p_j}$ in the target image $\mathbf{I}^{\mathbf{ref}}$ is set as the sum of a color distance and a positional distance:

$$c(i,j) = \lambda(\mathbf{c_i} - \mathbf{I}^{\mathbf{ref}}(\mathbf{p_j}))^2 + (1-\lambda)(\mathbf{p_i} - \mathbf{p_j})^2, \tag{13}$$

where $\mathbf{c_i} = \mathbf{I}(\mathbf{p_i})$ is the color of the rendered image at pixel $\mathbf{p_i}$, $\mathbf{I}^{\mathbf{ref}}(\mathbf{p_j})$ denotes the color of the target image at pixel $\mathbf{p_j}$, and λ is a balancing weight controlling the relative contributions between color and position. We set $\lambda = 0.5$ in our experiments.

In our scenario, the transportation matrix T_{ij} actually denotes a one-on-one mapping between pixels in the rendered and target images. Note that the pixel matching between rendered and target images could be also viewed as a matching between point proxies in the rendered image and pixels in the target image, since $\mathbf{c_i}$ and $\mathbf{p_i}$ could also denote the color and screen space position of the point proxy of the i-th pixel.

After the pixel matching is obtained, our loss function could be viewed as the sum of color and positional distances (Eq. 13) between matched pixels. Different from traditional L2 loss which is computed on aligned pixel locations, our loss function is able to handle global and long-range object correspondences.

4.3.2 Derivative of loss function. The above optimal transport based loss function (Eq. 12) can be easily differentiated by fixing the pixel matching (i.e., T_{ij} in the equation). Note that the pixel color and position in the target image ($\mathbf{I}^{\mathbf{ref}}(\mathbf{p_j})$ and $\mathbf{p_j}$) are also fixed. So that we could directly compute the derivative of the loss L with respect to both color $\mathbf{c_i}$ and screen space position $\mathbf{p_i}$ of point proxies according to Eq. 12, which is:

$$\left[\frac{\partial L}{\partial \mathbf{c_1}}, \frac{\partial L}{\partial \mathbf{p_1}}, \cdots, \frac{\partial L}{\partial \mathbf{c_N}}, \frac{\partial L}{\partial \mathbf{p_N}}\right]. \tag{14}$$

The final derivative with respect to a scene parameter could be computed through the chain rule:

$$\frac{\partial L}{\partial \theta} = \left[\frac{\partial L}{\partial \mathbf{c_1}}, \frac{\partial L}{\partial \mathbf{p_1}}, \cdots, \frac{\partial L}{\partial \mathbf{c_N}}, \frac{\partial L}{\partial \mathbf{p_N}}\right]^T \cdot \left[\frac{D\mathbf{c_1}}{D\theta}, \frac{\partial \mathbf{p_1}}{\partial \theta}, \cdots, \frac{D\mathbf{c_N}}{D\theta}, \frac{\partial \mathbf{p_N}}{\partial \theta}\right], \tag{15}$$

where the second term includes the RGBXY derivatives of all point proxies (Eq. 6). Imagine that, the purpose of the above derivatives would not only push the screen space position of the point proxy to be closer to that of the matched pixel, but also enforce its color closer to the matched pixel's color.

4.3.3 Approximation using Sinkhorn divergences. While our loss function in Eq. 12 and its derivatives could be directly computed using standard optimal transport algorithms [Peyré et al. 2019], however, it would be very slow to find the exact optimal solution. Instead, we utilize an approximated but rather efficient algorithm for optimal transport using Sinkhorn divergences [Cuturi 2013; Feydy et al. 2019]. It has a parameter ϵ to control its accuracy. Smaller values of ϵ lead to higher accuracy. We set $\epsilon = 0.01$ in our experiments.

4.3.4 An explanatory example. Let us look at a simple example to see why our method works. Fig. 3 shows a white rectangle with a single scene parameter θ which controls the amount of translation in the x-direction. We would like to move it from left (initial location) to right (target location), while there is no overlapping between the initial rectangle and the target rectangle. Blurred or anti-aliased input images are used to compute the standard color derivatives in order to prevent very large or infinite values.

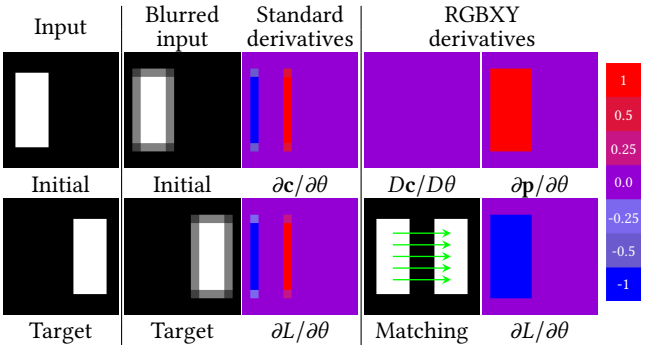


Fig. 3. An explaining example of comparing standard color derivatives and our RGBXY derivatives. The single scene parameter θ controls the amount of translation in the x-direction.

We could draw some useful observations from the results in Fig. 3. First, the standard derivatives ($\partial \mathbf{c}/\partial\theta$) are sparse — they are non-zero only near object boundaries, while our positional derivatives ($\partial \mathbf{p}/\partial\theta$) are much denser — having non-zero values in the whole interior area. This shows that our RGBXY derivatives could be more stable and useful. Second, by carefully looking at the map of loss derivatives ($\partial L/\partial\theta$) computed from standard derivatives, we could find that it is positive on one side and negative on the other side, probably leading to an overall zero-valued loss derivatives. The location of the rectangle would easily get stuck during optimization. In contrast, the map of loss derivatives in our method has dense and consistent values, and could effectively drive the rectangle move to the target location.

How to efficiently and effectively deal with object boundaries has been recognized as an open problem in differentiable rendering [Kato et al. 2020b]. The difficulty lies in the computation of the standard derivatives at edge boundaries which may involve very large values or infinitely-valued Dirac delta functions. While a lot of progress has been achieved by spatial blurring [Laine et al. 2020; Liu et al. 2019; Ravi et al. 2020], anti-aliasing [Laine et al. 2020], edge sampling [Li et al. 2018] or reparameterization [Bangaru et al. 2020; Loubet et al. 2019] however, due to intrinsic limitations of standard derivatives, i.e., its sparsity and locality, it is still not robust in handling scenes when the initial and target transformations differ a lot.

In contrast, we provide a simple and elegant solution for dealing with object boundaries. As shown in the above example, our method generates dense derivatives in interior regions and relies much less on boundary regions compared to traditional differentiable rendering methods. In other words, we can compute a dense, stable derivative and avoid computing problematic standard derivatives at object boundaries, i.e., having very large or infinite values.

5 OPTIMIZATION STRATEGIES AND IMPLEMENTATION

5.1 Optimization Strategies

Given a scene with parameters to be optimized, we could use optimizers like stochastic gradient descent or more sophisticated ones such as Adam optimizer [Kingma and Ba 2014]. In our experiments, we find that the Adam optimizer is less sensitive to parameter scales compared to stochastic gradient descent, hence, we always use the Adam optimizer. Compared with previous works, we need to change how the gradient is computed at each iteration, i.e., replacing the standard color derivatives (Eq. 3 and Eq. 5) with our RGBXY derivatives (Eq. 6 and Eq. 15).

Below, we further introduce two simple strategies used in the optimization process, which lead to higher running speed and better convergence.

5.1.1 Compute matching at intervals. As mentioned, our optimal transport based loss function requires computing pixel matching, i.e., the transportation matrix T in Eq. 12, as well as computing the color and positional distances between matched pixels (Eq. 13). On one hand, computing pixel matching is costly and is the bottleneck in computation. On the other hand, there are only subtle changes in pixel matching within adjacent iteration steps. Hence, instead of computing the matching at each step, we opt to compute it at intervals (i.e., once every $K+1$ steps, and we use $K=5$) for higher speed without hurting accuracy.

5.1.2 Hybrid gradients. Our RGBXY derivatives are highly effective in capturing global and long-range influence, however, it would be less effective when initial and target shapes are already aligned very well. This is probably due to that our used approximated optimal transport method may lead to inaccurate matching when the initial and target shapes are already very close. Motivated by this, we additionally introduce a hybrid strategy that combines RGBXY derivatives and standard derivatives, leveraging the advantages of both. During optimization, we use RGBXY derivatives (Eq. 15) at the beginning and switch to standard derivatives (Eq. 5) at the last. In our implementation, we simply employ a 3:1 split — using RGBXY derivatives for the first 75% iterations and switching to standard derivatives for the last 25% iterations.

We will validate the above two strategies later in Sec. 6.1.

5.2 Implementation Details

Our method is very simple to implement on top of existing differential rendering frameworks. We have implemented our method on three differentiable rendering frameworks, including Nvdiffrast [Laine et al. 2020], PyTorch3D [Ravi et al. 2020], and JRender [Hu et al. 2020]. In our implementation, no soft blending along the z-axis or silhouette blending on pixels is employed.

5.2.1 Barycentric coordinates. All differentiable rendering frameworks provide functions to fetch the barycentric coordinates of a pixel with respect to its covering triangle. Care must be taken when dealing with the barycentric coordinates of point proxies. Since the point proxies represent the underlying geometric points, hence, their barycentric coordinates should not change by definition, which means that their gradient should always be zero.

5.2.2 Background pixels. Background pixels, i.e., those pixels which do not cover any geometries, are still involved in the computation of optimal transport based pixel matching. However, since they are not covered by any triangles and are not associated with any point proxies, they will not contribute to the final gradients. Nevertheless, if a foreground mask is additionally provided together with the target image, we could safely exclude the background pixels in computations.

5.2.3 Antialiased pixels. For antialiased pixels at or near object boundaries, they may be partially covered by multiple triangles. The more sophisticated way will need to record all of its covering triangles, track and compute the RGBXY derivatives for all underlying geometric points on all covering triangles, however, this will lead to unnecessary implementation complexity. Instead, we always associate such pixel with one underlying geometric point, i.e., the point which pixel center hits. As explained in Sec. 4.3.4, our derivatives are mainly contributed from interior regions rather than from boundaries, the above solution works well without sacrificing accuracy.

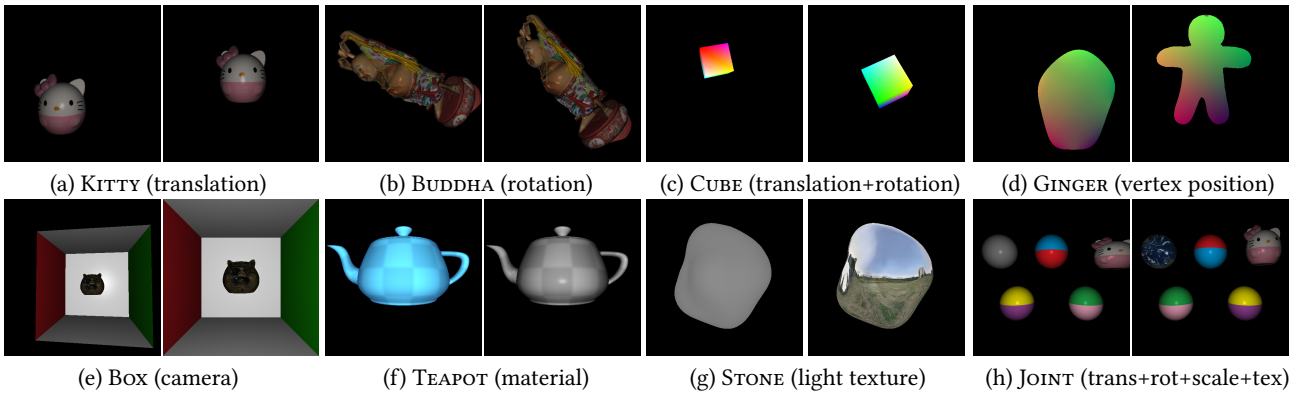
5.2.4 Implementation of approximated optimal transport. We opt for Sinkhorn divergences for an approximated but efficient optimal transport algorithm. Specifically, we directly use the GPU implementation provided by Feydy et al. [2019] with parameter $\epsilon = 0.01$, which corresponds to a highly accurate matching. The time complexity of the algorithm is $O(NM)$, where N, M is the total pixel amount in two images. Processing a matching between two 128×128 images typically takes about 0.05 seconds on an NVIDIA RTX 3090 GPU. The processing time increases to 0.43 seconds for two 256×256 images, and to 5.72s for two 512×512 images.

6 EXPERIMENTS

All experiments are performed on a PC with an NVIDIA RTX 3090 GPU with 24GB memory. Since the Nvdiffrast framework is faster, we always use our Nvdiffrast based implementation. We leverage Adam optimizer [Kingma and Ba 2014] using default parameters of $\alpha = 0.02, \beta_1 = 0.9$ and $\beta_2 = 0.999$, with the learning rate decayed by $0.999\times$ after each iteration.

6.1 Evaluation

To evaluate the robustness of our method, we have collected 8 scenes as shown in Fig. 4. Each scene has one or multiple types of parameters to optimize, including translation vector, rotation angle, vertex position, camera pose, material, environment light, and textures. The tested scene parameters include both geometric and non-geometric parameters, as well as combinations of parameters. The number of views used for each scene is given in Table 1, ranging from 1 view to 6 views. For scenes Kitty, Buddha, Cube, Box, and Teapot, we randomly generate 10 groups of initial scene parameters. For the 3 other scenes, since randomly initialized parameters may lead to meaningless configurations, e.g., self-intersection meshes, we manually provide 1 group of initial scene parameters.



(a) Kitty (translation) (b) Buddha (rotation) (c) Cube (translation+rotation) (d) Ginger (vertex position)

(e) Box (camera) (f) Teapot (material) (g) Stone (light texture) (h) Joint (trans+rot+scale+tex)

Fig. 4. The 8 scenes for evaluation. For each scene, we provide the initial image on the left and the target image on the right, from one of the views (if having multiple views). The scene parameters to be optimized are given inside the bracket. (a) Kitty optimizing for a translation vector; (b) Buddha optimizing for a rotation angle; (c) Cube optimizing for a translation vector and a rotation angle; (d) Ginger optimizing for all vertex positions; (e) Box optimizing for the camera pose; (f) Teapot optimizing for diffuse and specular parameters; (g) Stone optimizing for the texture representing environment light; (h) Joint optimizing for several types of scene parameters, including translation, rotation, scale, and texture.

Table 1. Evaluation results on 8 scenes, including (a) alternative matching methods, including optical flow and bipartite graph matching; (b) different values of parameter λ; (c) different values of matching interval K; (d) different optimization strategies including a randomly combination strategy, the hybrid strategy, and the basic strategy using only RGBXY derivatives; (e) baseline methods including Nvdiffrast variants with resolution 128×128, with resolution 1024×1024, and with two multi-scale schemes, respectively.

Scene		Kitty		Buddha		Cube		Ginger		Box		Teapot		Outdoor		Joint		Speed (It/sec)
Optimize Parameter		Translation		Rotation		Translation Rotation		Vertex Position		Translation View Direction		Diffuse Specular		Envlight Cubemap		Trans+Rot+ Scale+Texture		
Num of views		6		6		4		1		1		6		6		1		
Metric		MAE	PSNR	MAE	PSNR	MAE	PSNR	MAE	PSNR	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	
(a) matching algo.(K = 5,basic)	op.flow	0.010	34.25	**0.342**	22.41	1.020	17.33	0.477	18.93	12.77	0.530	17.43	0.905	18.97	0.813	18.87	0.840	19.49
	bipart.	0.013	33.68	0.433	23.09	0.171	24.54	0.598	14.33	14.25	0.620	**53.10**	0.997	18.95	0.800	16.94	0.540	87.93
(b) param λ (K = 5,basic)	λ = 0.01	0.639	19.16	0.970	18.06	0.920	16.98	0.535	14.80	14.71	0.783	44.86	0.997	22.28	0.905	16.56	0.731	62.68
	λ = 0.1	0.332	20.74	0.775	19.01	0.007	**38.29**	0.357	25.18	16.85	0.840	35.95	0.986	18.66	0.828	16.56	0.731	62.56
	λ = 0.9	**0.001**	45.63	0.442	22.55	0.063	27.28	0.306	29.68	22.89	0.930	32.39	0.961	15.89	0.741	25.25	0.886	60.58
	λ = 0.99	**0.001**	**47.38**	0.429	22.73	0.066	27.11	0.309	29.14	23.69	0.940	33.27	0.969	16.79	0.736	25.89	0.907	54.50
(c) match interval K (λ = 0.5,basic)	K = 0	0.002	42.04	0.457	24.50	0.066	27.35	0.302	31.26	22.76	0.932	33.88	0.967	17.04	0.779	22.84	0.843	15.34
	K = 1	0.003	41.98	0.460	24.54	0.062	27.59	0.308	28.28	22.34	0.923	33.91	0.967	17.10	0.781	22.05	0.827	27.94
	K = 10	0.002	42.25	0.489	21.66	0.047	29.06	0.300	30.20	22.18	0.920	33.42	0.967	17.19	0.785	23.96	0.865	88.81
	K = 50	0.011	33.88	0.708	18.81	0.048	28.91	**0.297**	31.34	21.29	0.909	31.66	0.962	16.63	0.793	20.15	0.779	148.28
(d) optim strategy K = 5,λ = 0.5)	rand	0.419	24.32	0.806	20.49	1.666	18.77	0.537	25.85	15.56	0.820	49.59	**0.999**	54.02	**1.000**	21.69	0.898	90.15
	hybrid	0.003	41.53	0.467	24.53	**0.006**	37.87	0.305	**35.14**	**25.28**	0.952	49.08	**0.999**	51.21	**1.000**	**44.19**	**0.996**	74.08
	basic	0.002	42.41	0.480	22.88	0.052	28.46	0.304	28.69	22.69	0.931	33.86	0.968	16.85	0.779	24.00	0.867	61.96
(e) baseline	nvdf.res.128	0.487	24.47	0.857	21.02	1.887	18.98	0.560	24.68	14.54	0.796	50.16	**0.999**	**54.50**	**1.000**	21.69	0.899	**173.20**
	nvdf.res.1024	0.491	24.56	0.751	20.69	1.878	19.11	0.551	28.14	18.02	0.935	39.18	0.989	53.25	0.999	23.36	0.950	112.88
	nvdf.mul.res	0.404	24.28	0.772	19.47	2.217	17.68	0.653	22.24	13.38	0.828	41.81	0.987	44.42	0.986	21.38	0.922	146.35
	nvdf.down.res	0.635	24.60	0.449	**25.71**	1.359	20.00	0.581	29.06	18.91	**0.953**	38.05	0.970	51.49	0.998	23.36	0.950	109.56

Then, we conduct a series of evaluations to verify our design choices and parameter settings. Note that when we evaluate on a specific setting (parameter), other parameters are retained to be the same as our final method. In each test, we use a rendering resolution of 128×128 and optimize for 1000 iterations.

We provide an evaluation with both 3D and 2D metrics, i.e., a 3D error metric MAE which measures the mean absolute error between vertices of the optimized and target meshes, and 2D error metrics including PSNR, SSIM [Wang et al. 2003], RMSE and a perceptual score LPIPS [Zhang et al. 2018]. The averaged scores are given in Table 1. Due to space limitations, we only show MAE and PSNR scores for scenes with only geometric changes and show PSNR and SSIM for other scenes. Full results can be found in the supplemental document.

6.1.1 Optimal transport vs other pixel matching methods. As discussed in Sec. 4.3, our method uses approximated optimal transport to capture global and long-range correspondences. Other techniques could also serve the same purpose, such as optical flow or bipartite graph matching. For optical flow, we have selected a state-of-the-art method [Truong et al. 2022] to compare with. For bipartite graph matching, we use the implementation [Crouse 2016] provided in the *SciPy* library. Table 1 (a) shows the scores by replacing optimal transport with optical flow and bipartite graph matching, respectively. Rendering resolution is set to 128×128 for optical flow and

is set to 32×32 for bipartite graph matching. A matching interval $K = 5$ is used for a trade-off between speed and quality. Their results are generally less robust than ours (Table 1 (d)): both of them leads to unsatisfactory results (PSNR < 20dB) on scenes Box and Ginger. Furthermore, optical flow [Truong et al. 2022] is slower than ours, while bipartite graph matching [Crouse 2016] does not scale well with image size, i.e., it is significantly slower if rendering resolution is set to 64×64. Overall, optimal transport is the best choice.

6.1.2 The weight parameter λ in Eq. 13. The weight parameter λ is used to control the relative contributions between colors and positions in computing the loss. Generally, higher values of λ lead to longer-range capturing while lower values λ lead to closer behavior to traditional differentiable rendering methods. The results shown in Table 1 (b) are consistent with our expectations. Small values (i.e., $\lambda < 0.5$) generally perform better on scenes that do not optimize for geometric parameters, while large values are on the opposite. Overall, $\lambda = 0.5$ (Table 1 (d)) is a suitable choice.

6.1.3 The matching interval K. As described in Sec. 5.1.1, during optimization, instead of computing pixel matching at every iteration, computing it at intervals could accelerate the optimization process. Here, we evaluate different choices of matching intervals, including $K = 0, 1, 5, 10$, and 50, where $K = 0$ denotes computing matching at every iteration. As shown in Table. 1 (c) and (d), setting $K = 5$ or $K = 10$ generally lead to similar result qualities compared to setting $K = 0$, while increasing the running speed by 4× to 5×. Hence, we set $K = 5$ by default.

6.1.4 The hybrid strategy. As described in Sec. 5.1.2, we additionally introduce a hybrid optimizing strategy, which uses RGBXY derivatives for the first 75% iterations and switches to standard derivatives for the last 25% iterations. We also test a randomized strategy for combining RGBXY derivatives and standard derivatives, i.e., randomly selecting one of them at each iteration. As shown in Table. 1 (d), the hybrid strategy performs surprisingly well in all scenes while the random strategy does not show benefits. It agrees with our expectations: RGBXY derivatives are effective at capturing global and long-range relationships hence they should be used in the beginning, and using standard derivatives in the end could help to finetune parameters for more accurate reconstruction.

6.1.5 Comparison with baselines. We also include 4 versions of Nvdiffrast [Laine et al. 2020] as baselines for comparison, as shown in Table. 1 (e), including two without and two with multi-scale schemes. The first one uses a rendering resolution of 128×128. The second one uses a rendering resolution 1024×1024. The third one (with name 'nvdf.mul.res' in the table) uses a multi-scale scheme which progressively increases rendering resolution from 8×8 to 1024×1024. The fourth one (with name 'nvdf.down.res' in the table) also employs a multi-scale scheme. It always renders images with a resolution of 1024×1024, however, it progressively uses downsampled rendered images with resolution from 8×8 to 1024×1024 for computing loss functions. While using multi-scale schemes or using a higher resolution (i.e., 1024×1024) could improve upon using a low resolution (i.e., 128×128), they still perform generally less successful than our method using a low resolution (i.e., 128×128), due to the less effectiveness of standard derivatives compared to RGBXY derivatives.

6.1.6 Overall observations from evaluation results. First, the results verify that our basic method (i.e., the one without hybrid strategy) is highly effective in dealing with geometric scene parameters, especially when the initial rendered image and target image has large long-range differences. It is not surprising that we perform less effectively compared to Nvdiffrast on scenes Teapot and Stone which only optimize for materials and environment lights, respectively, since optimal transport will not have benefits if there are no geometric changes between initial and target states.

Second, our hybrid approach is even better and more robust — it can increase quality in almost all tested examples, and can be robustly used for all types of scene parameters, including scenes Teapot and Stone.

Overall, we suggest the following default setting: using weight $\lambda = 0.5$, matching interval $K = 5$, and with hybrid strategy enabled.

6.2 Applications

To demonstrate the practicability, we present several inverse rendering applications of our method, including (1) furniture layout adjustment, (2) human pose fitting and (3) facial expression reconstruction. For each application, we compare results optimized with our method against those optimized with standard L2 loss using both Nvdiffrast [Laine et al. 2020] and PyTorch3D [Ravi et al. 2020]. For Nvdiffrast, we compute image space L2 loss between antialiased rendering images and target references. For PyTorch3D, we set the blur radius to 10^{-4} and render 50 softened triangles at maximum along each ray for alpha blending, and the L2 loss is computed between target references and the aggregated rendering images. Since the soft boundary is an important feature for PyTorch3D, we also add another baseline with L2 loss computed on silhouette masks. Furniture layout adjustment and facial expression reconstruction use only one view and a rendering resolution of 256×256 for optimization, while human pose fitting uses 6 views and a rendering resolution of 128×128. We run each method for 1000 iterations.

6.2.1 Furniture layout adjustment. Indoor digital 3D scenes exhibit rich details including varying materials, intricate layouts, complex shapes and decorations. In this experiment, we randomly select 50 scenes from the 3D-Front indoor scene dataset [Fu et al. 2021] and focus on automatically aligning the furniture layout through differentiable rendering. Given a target image with desired furniture layout, we initialize the optimization with randomly placed furniture locations in the room. Specifically, for each 3D furniture model, we optimize for its translation on X-Z plane and rotation along Y-axis. The visual results of two scenes are shown in Fig. 1. PyTorch3D uses blurred geometry borders to compute boundary gradients, leading to significant bias when scene complexity is high. Nvdiffrast uses analytic antialiasing to produce more reliable gradients, but often got stuck in local minima due to the lack of global and long-range gradients. In contrast, our method outperforms the baselines regarding both 3D and 2D metrics, matching the target layout well.

6.2.2 Human pose fitting. Human pose estimation is an active and important research topic in computer vision. A commonly used

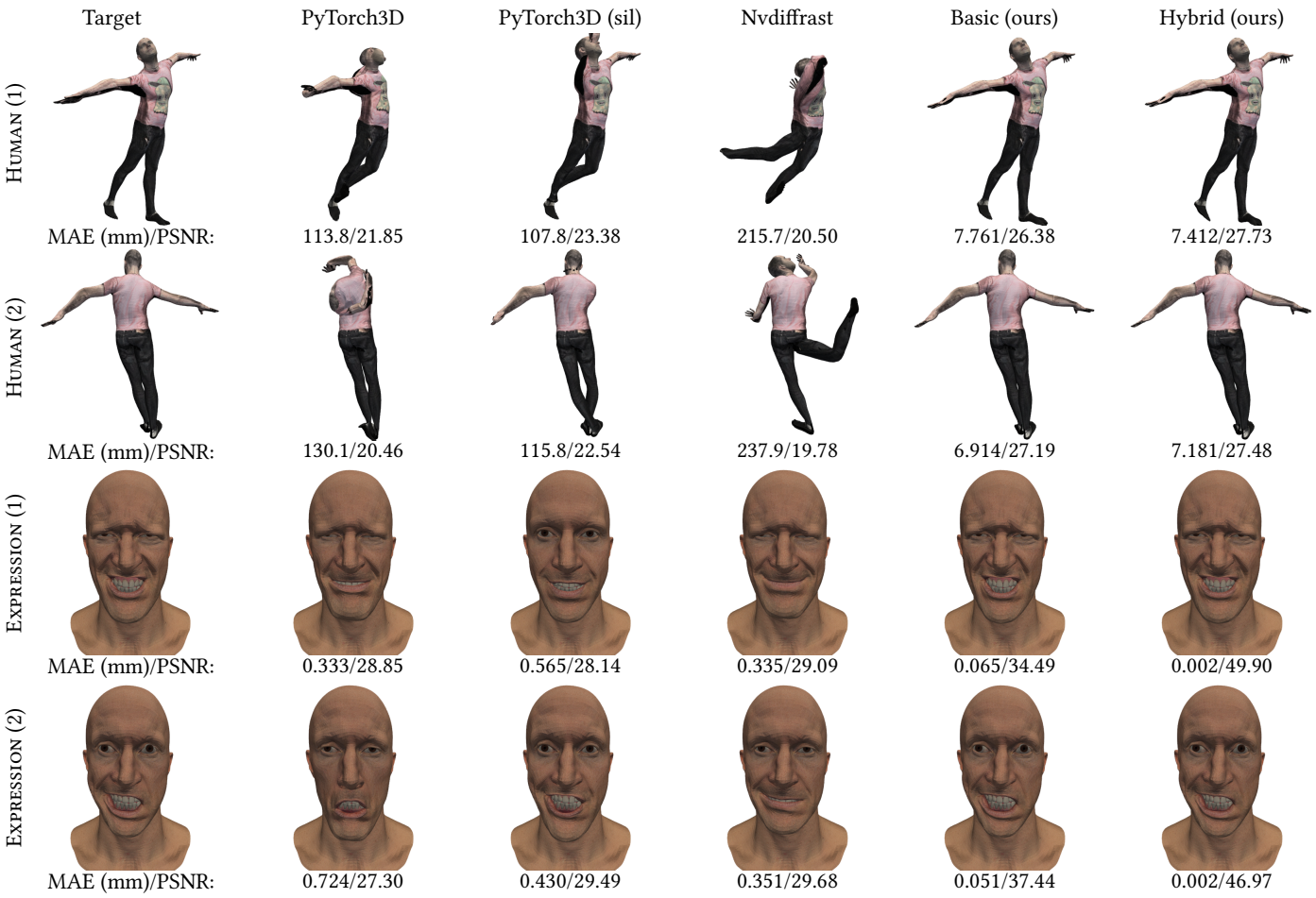


Fig. 5. Visual results of applications, including human pose fitting (top two rows) and facial expression reconstruction (bottom two rows). Our methods consistently outperform the baselines with a large gap.

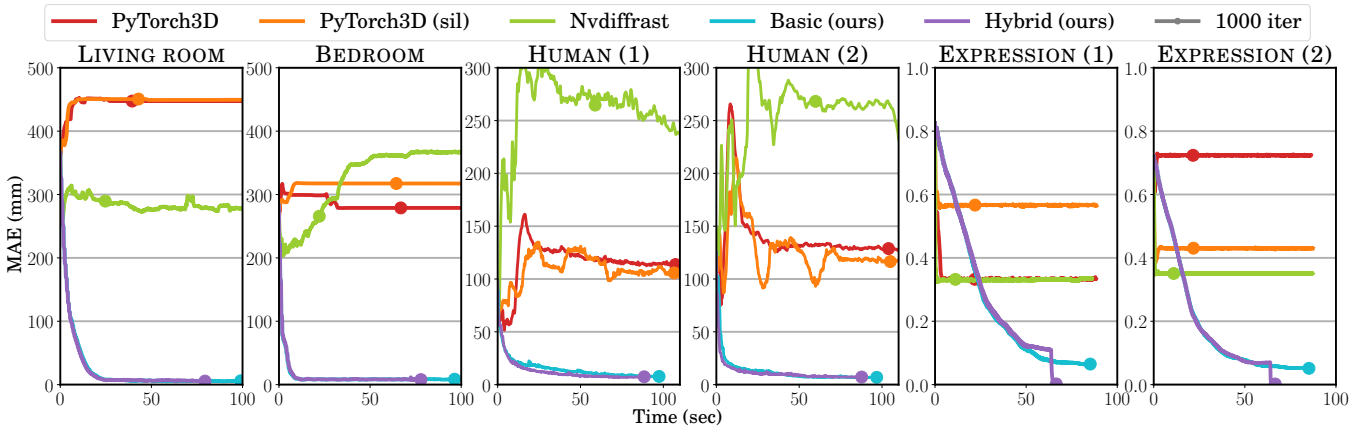


Fig. 6. Error curve. We show how the MAE metric changes with time during the optimization for all 6 scenes in Fig. 1 and Fig. 5.

parameterization of human model is Skinned Multi-Person Linear Model (SMPL) [Loper et al. 2015], which takes 24 rotation angles (around joints) and 10 shape parameters as input to generate a human mesh. In this experiment, we aim to fit the SMPL parameters to match a given reference image that contains the target human pose. We generated 50 target poses by assigning random rotation angles between $[-0.4, 0.4]$ and rendered 6 views of the human models. We set all SMPL parameters to zero as initialization. The visual comparisons are provided in Fig. 5 top, demonstrating the superiority of our method.

6.2.3 Facial expression reconstruction. High-fidelity reconstruction of facial expressions is another important topic in many areas such as non-rigid registration and deformation. Here, we use Blendshape [Lewis et al. 2014] to fit a target expression. We have one base model b_0 with a neutral expression and $n = 12$ models with different expressions $b_1, b_2 \dots b_n$. Given different combination of weights $w_1, w_2 \dots w_n$, we can generate face models with various expressions via $b = b_0 + \sum_{i=1}^{n} w_i(b_i - b_0)$. We generated 50 target expressions by assigning random weights between $[0, 0.5]$. Again, we set weights to zero (i.e., neutral expression) as the initialization for all methods. As shown in Fig. 5 bottom, our methods consistently outperform the baselines with a large gap.

In Fig. 6, we further visualize how the reconstruction error (i.e., MAE) changes with time during the optimization for all 6 scenes shown in Fig. 1 and Fig. 5. While PyTorch3D and Nvdiffrast still exhibit large errors after 1000 iterations, our method converges quickly on all scenes. Noticing the error curves for Expression (1) and Expression (2), we could find that our hybrid strategy further improves converge during the last iteration steps.

More visual results are given in the supplemental document.

6.3 Discussion and Analysis

6.3.1 Relationship with Lagrangian fluid simulation. In a sense, traditional differentiable rendering methods might be considered as adopting Eulerian view since they always compute derivatives at fixed screen space locations, while our method is from Lagrangian view since we operate on point proxies. Such analogy is different from the fluid simulation counterparts. First, in fluid simulation, Lagrangian methods are solving the same mathematical problem (i.e., the same Navier-Stokes equation) as Eulerian methods, however, our method is solving a different optimization problem from traditional differentiable rendering methods since we use a different loss function. Second, we use RGBXY derivatives by combining material derivatives and positional derivatives which is different from the way how Lagrangian derivatives are used in fluid simulation.

6.3.2 Visualization of matching results. In Fig. 7, we show visualizations of our optimal transport based pixel matching on three examples. For each example, we provide the target reference image, the rendered image, the XY positions of matched pixels, and the reconstructed target image from the rendered image. From the results, we could find that large-scale transformations of objects can be generally well captured by optimal transport. However, we could also notice that many object details are not correctly matched and there are many noises in the reconstructed target image. This also suggests that the hybrid strategy that switches to standard derivatives in the end would be useful.

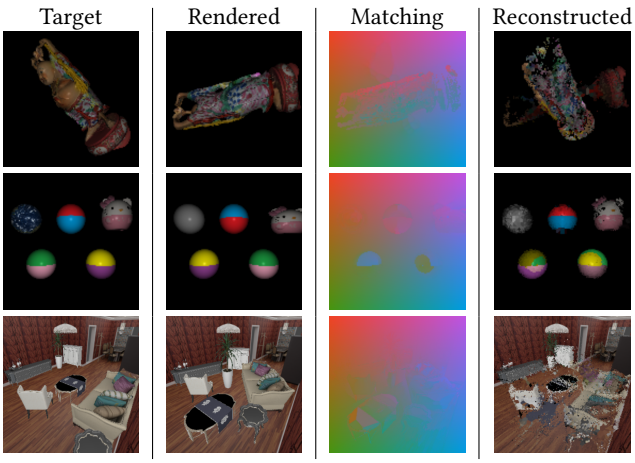


Fig. 7. Visualization of our optimal transport based pixel matching. We provide three examples, including scenes Buddha (Fig. 4) and Joint (Fig. 4) and Living Room (Fig. 1). For each example, from left to right, we provide the target reference image, the rendered image, the XY positions of matched pixels, and the reconstructed target image using the color of matched pixels from the rendered image.

6.3.3 Limitations. Our method has several limitations. First, if the movement of an appearance effect is not consistent with the movement of the underlying geometry point, our method would probably fail, e.g., moving shadows or moving reflection highlights caused by the movement of light sources. An example of moving reflection highlights is shown in Fig. 8 top , where we optimize for the position of the point light source. While our method could successfully find the matching of highlight pixels between the initial and target images, it fails to recover the position of light sources.

Second, our method relies on optimal transport to obtain pixel matching. The accuracy of pixel matching will certainly affect the quality of our results. Such an example is shown in Fig. 8 bottom, where we would like to rotate the textured square by 90°. However, since the texture looks rather similar everywhere, optimal transport cannot produce the correct matching.

It is necessary to note that traditional differentiable rendering frameworks (i.e., PyTorch3D and Nvdiffrast) cannot robustly handle the two scenes in Fig. 8, either.

7 CONCLUSION AND FUTURE WORK

We proposed a novel differentiable rendering method to improve the robustness of differentiable rendering. Instead of computing image derivatives locally on fixed screen space locations, we associate pixels with point proxies and track the movement of the point proxies. By leveraging RGBXY derivatives and optimal transport based loss functions, our method is able to capture global and long-range object correspondences. We further proposed a hybrid gradient

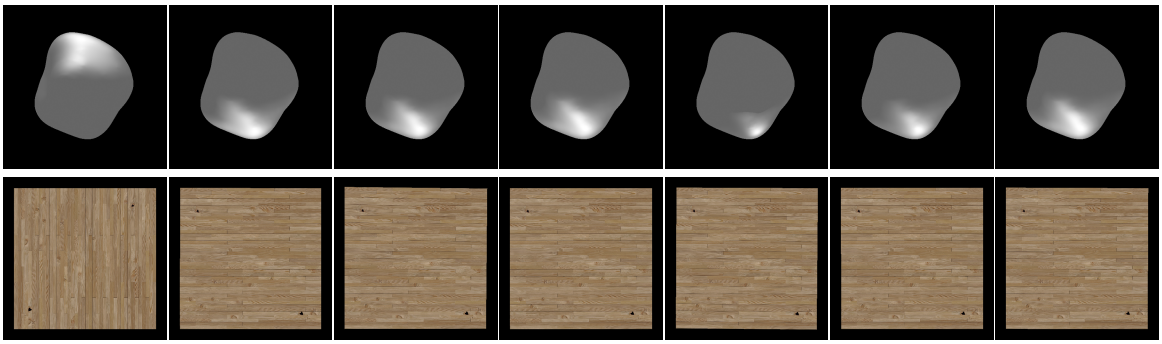


Fig. 8. Two failure cases of our method. The top row shows a scene where we would like to optimize for the position of the point light source. The bottom row shows a scene where we would like to rotate the textured square by 90°.

strategy with better convergence properties. Experiments show that our method is more robust and effective than existing differentiable rendering methods in various inverse rendering applications.

In the future, first, we would like to extend our method to support ray tracing in order to handle more complex global illumination effects. This might be achieved by tracking a full light path, rather than the first hit geometry point. Second, supporting implicit representations such as SDF or NeRF [Mildenhall et al. 2020] is also an interesting future works. Besides, we believe the limitations mentioned above (e.g., moving shadows or reflection highlights, inaccurate pixel matching) would also stimulate future works along this direction. A possible way to design more sophisticated matching algorithms would be taking 3D geometry into consideration or dynamically balancing the weights of color and positional distances.

ACKNOWLEDGMENTS

We would like to thank all reviewers for their helpful comments. This work is supported by the National Natural Science Foundation of China (Project Number: 61932003). Ling-Qi Yan is supported by gift funds from Adobe, Dimension 5 and XVerse.

REFERENCES

Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. 2020. Unbiased warped-area sampling for differentiable rendering. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–18.

Nicolas Bonneel and David Coeurjolly. 2019. SPOT: Sliced Partial Optimal Transport. *ACM Transactions on Graphics (SIGGRAPH)* 38, 4 (2019).

Nicolas Bonneel, Gabriel Peyré, and Marco Cuturi. 2016. Wasserstein barycentric coordinates: histogram regression using optimal transport. *ACM Trans. Graph.* 35, 4 (2016), 71–1.

Nicolas Bonneel, Michiel Van De Panne, Sylvain Paris, and Wolfgang Heidrich. 2011. Displacement interpolation using Lagrangian mass transport. In *Proceedings of the 2011 SIGGRAPH Asia conference*. 1–12.

Robert Bridson. 2008. *Fluid Simulation for Computer Graphics*. A K Peters/CRC Press.

Guangyan Cai, Kai Yan, Zhao Dong, Ioannis Gkioulekas, and Shuang Zhao. 2022. Physics-Based Inverse Rendering using Combined Implicit and Explicit Geometries. *arXiv preprint arXiv:2205.01242* (2022).

Chengqian Che, Fujun Luan, Shuang Zhao, Kavita Bala, and Ioannis Gkioulekas. 2020. Towards learning-based inverse subsurface scattering. In *2020 IEEE International Conference on Computational Photography (ICCP)*. IEEE, 1–12.

David F. Crouse. 2016. On implementing 2D rectangular assignment algorithms. *IEEE Trans. Aerospace Electron. Systems* 52, 4 (2016), 1679–1696. https://doi.org/10.1109/TAES.2016.140952

Marco Cuturi. 2013. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems* 26 (2013).

Xi Deng, Fujun Luan, Bruce Walter, Kavita Bala, and Steve Marschner. 2022. Reconstructing Translucent Objects using Differentiable Rendering. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*. 1–10.

Mathieu Desbrun and Marie-Paule Gascuel. 1996. Smoothed Particles: A New Paradigm for Animating Highly Deformable Bodies. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96* (Poitiers, France). Springer-Verlag, Berlin, Heidelberg, 61–76.

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual Simulation of Smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 15–22. https://doi.org/10.1145/383259.383260

Jean Feydy, Thibault Séjourné, François-Xavier Vialard, Shun-ichi Amari, Alain Trouve, and Gabriel Peyré. 2019. Interpolating between Optimal Transport and MMD using Sinkhorn Divergences. In *The 22nd International Conference on Artificial Intelligence and Statistics*. 2681–2690.

Huan Fu, Bowen Cai, Lin Gao, Ling-Xiao Zhang, Jiaming Wang, Cao Li, Qixun Zeng, Chengyue Sun, Rongfei Jia, Binqiang Zhao, et al. 2021. 3D-Front: 3D Furnished Rooms with Layouts and Semantics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 10933–10942.

Duan Gao, Xiao Li, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. 2019. Deep inverse rendering for high-resolution SVBRDF estimation from an arbitrary number of images. *ACM Trans. Graph.* 38, 4 (2019), 134–1.

Yu Guo, Cameron Smith, Miloš Hašan, Kalyan Sunkavalli, and Shuang Zhao. 2020. MaterialGAN: reflectance capture using a generative SVBRDF model. *arXiv preprint arXiv:2010.00114* (2020).

Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. 2021. Appearance-Driven Automatic 3D Model Simplification. In *Eurographics Symposium on Rendering*.

Shi-Min Hu, Dun Liang, Guo-Ye Yang, Guo-Wei Yang, and Wen-Yang Zhou. 2020. Jittor: a novel deep learning framework with meta-operators and unified graph execution. *Science China Information Sciences* 63, 222103 (2020), 1–222103. https://github.com/Jittor/jrender

E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. 2017. FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. http://lmb.informatik.uni-freiburg.de//Publications/2017/IMKDB17

Leonid V Kantorovich. 1942. On the translocation of masses. In *Dokl. Akad. Nauk. USSR (NS)*, Vol. 37. 199–201.

Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. 2020a. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057* (2020).

Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. 2020b. Differentiable Rendering: A Survey. https://doi.org/10.48550/ARXIV.2006.12057

Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. 2018. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3907–3916.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

H. W. Kuhn and Bryn Yaw. 1955. The Hungarian method for the assignment problem. *Naval Res. Logist. Quart* (1955), 83–97.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. 2020. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–14.

J. P. Lewis, Ken Anjyo, Taehyun Rhee, Mengjie Zhang, Fred Pighin, and Zhigang Deng. 2014. Practice and Theory of Blendshape Facial Models. In *Eurographics 2014 - State of the Art Reports*, Sylvain Lefebvre and Michela Spagnuolo (Eds.). The Eurographics Association. https://doi.org/10.2312/egst.20141042

Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–11.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. 2019. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7708–7717.

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. 2015. SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 34, 6 (Oct. 2015), 248:1–248:16.

Matthew M Loper and Michael J Black. 2014. OpenDR: An approximate differentiable renderer. In *European Conference on Computer Vision*. Springer, 154–169.

Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. 2019. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–14.

Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. 2021. Unified shape and svbrdf recovery using differentiable monte carlo rendering. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 101–113.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*. Springer, 405–421.

Gaspard Monge. 1781. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris* (1781).

Matthias Müller, David Charypar, and Markus Gross. 2003. Particle-Based Fluid Simulation for Interactive Applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '03)*. Eurographics Association, Goslar, DEU, 154–159.

Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. 2021. Extracting Triangular 3D Models, Materials, and Lighting From Images. *arXiv preprint arXiv:2111.12503* (2021).

Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. 2021. Large steps in inverse rendering of geometry. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–13.

Lois Paulin, Nicolas Bonneel, David Coeurjolly, Jean-Claude Iehl, Antoine Webanck, Mathieu Desbrun, and Victor Ostromoukhov. 2020. Sliced optimal transport sampling. *ACM Trans. Graph.* 39, 4 (2020), 99.

Gabriel Peyré, Marco Cuturi, et al. 2019. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning* 11, 5-6 (2019), 355–607.

Simon Premžoe, Tolga Tasdizen, James Bigler, Aaron Lefohn, and Ross T. Whitaker. 2003. Particle-Based Simulation of Fluids. *Computer Graphics Forum* 22, 3 (2003), 401–410. https://doi.org/10.1111/1467-8659.00687

Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. 2020. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501* (2020).

Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. 2020. MATch: Differentiable material graphs for procedural material capture. (2020).

Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. 2015. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (ToG)* 34, 4 (2015), 1–11.

Jos Stam. 1999. Stable Fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 121–128. https://doi.org/10.1145/311535.311548

Prune Truong, Martin Danelljan, Fisher Yu, and Luc Van Gool. 2022. Probabilistic Warp Consistency for Weakly-Supervised Semantic Correspondences. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*. https://arxiv.org/abs/2203.04279

Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2022. Differentiable signed distance function rendering. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–18.

Zhou Wang, Eero P Simoncelli, and Alan C Bovik. 2003. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, Vol. 2. Ieee, 1398–1402.

Cheng Zhang, Bailey Miller, Kan Yan, Ioannis Gkioulekas, and Shuang Zhao. 2020. Path-space differentiable rendering. *ACM transactions on graphics* 39, 4 (2020).

Cheng Zhang, Zihan Yu, and Shuang Zhao. 2021. Path-space differentiable rendering of participating media. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–15.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 586–595.

Shuang Zhao, Ioannis Gkioulekas, and Sai Bangaru. 2021. CVPR 2021 Tutorial on Physics-Based Differentiable Rendering. https://www.diff-render.org/.