CS598 Paper Survey: Differentiable Rendering

Hao-Yu Hsu, Yufeng Liu
{haoyuyh3,yufengl2}@illinois.edu

Abstract

Differentiable rendering (DR) has emerged as a crucial technique for integrating 3D scene understanding into neural networks, enabling end-to-end optimization of 3D scene parameters using 2D image data. It reduces the need for extensive 3D data collection and annotations. In this survey, we review the current state of DR algorithms, focusing on various 3D representations, including meshes, voxels, point clouds, and neural implicit representations. Additionally, we explore evaluation methods, downstream applications utilizing DR, and introduce several open-source DR libraries.

1. Introduction

In recent years, it has become evident that neural networks are effective for reasoning in both 2D and 3D environments. However, most methods for 3D estimation depend on supervised training and expensive annotations, making it difficult to gather comprehensive data about 3D observations. Consequently, there have been initiatives aimed at utilizing more readily available 2D data and various levels of supervision for 3D scene comprehension. One such approach involves integrating graphical rendering processes into neural network workflows, enabling the transformation and inclusion of 3D estimates with 2D image data.

Rendering, a crucial aspect of computer graphics, involves creating images of 3D scenes defined by their geometry, materials, lighting, and camera settings. This process of creating an image based on scene parameters is referred to as *forward rendering*. The inverse process of optimizing the scene parameters based on image gradients is referred to as *inverse rendering*. Because traditional forward rendering lacks a straightforward differentiation method, it is challenging to set up the inverse rendering process, and thus difficult to integrate into neural networks. Differentiable rendering (DR) comprises a set of techniques that address this integration for end-to-end optimization by providing useful gradients from the rendering process. By differentiating rendering, DR effectively connects 2D and 3D processing, allowing neural networks to optimize 3D scene parameters while working with 2D projections. This is done through backpropagation of gradients from the rendering output. Typically, a 3D self-supervised pipeline integrates a rendering layer with predicted scene parameters and applies a loss function between the rendered images and the input images. This process has wide-ranging applications, including 3D object reconstruction, human pose estimation, hand pose estimation, and face reconstruction.

In addition to making the traditional rendering pipeline differentiable, new methods of neural rendering have gained significant success in recent years. These methods are differentiable by construction and can thus be classified as a form of differentiable rendering. In contrast to traditional explicit representations of scene parameters, neural rendering methods use implicit representations.

Despite the various methods proposed, it is still not straightforward to decide which method to use in a certain scenario. This can be attributed to the following reasons:

- Rendering pipelines for different 3D representations differ significantly, necessitating different strategies to make the pipelines differentiable.
- It is unclear how recent neural network-based DR differs from traditional physics-based DR.

To address these issues, a comprehensive survey of the current state of DR is needed. In this work, we present an overview of current DR algorithms.

2. Algorithm

2.1. Problem Formulation

We begin by establishing a mathematical framework tailored to our needs. A rendering function R takes as input shape parameters Φ_s, camera parameters Φ_c, material parameters Φ_m, and lighting parameters Φ_l, producing either an RGB image I_c or a depth image I_d. We represent the inputs as $\Phi = \{\Phi_s, \Phi_m, \Phi_c, \Phi_l\}$ and the outputs as $I = \{I_c, I_d\}$. While a general formulation of differentiable rendering (DR) can include various types of additional input/output entities, this section focuses on the most common ones. The overview is illustrated in Fig. 1.

A differentiable renderer calculates the gradients of the output image concerning the input parameters $\frac{\partial I}{\partial \Phi}$ to optimize a specific objective (loss function). These gradient computations may be approximate, but they should be sufficiently accurate to convey meaningful information neces-

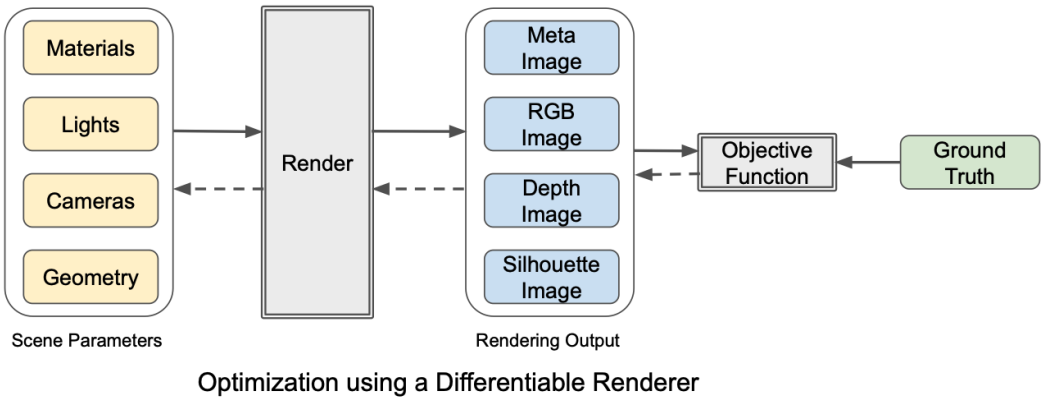

Figure 1. Schematic overview of optimization framework using a differentiable renderer.

sary for minimizing the objective function.

The inputs to a rendering function Φ, particularly the geometric parameters Φ_s, are key factors that distinguish different differentiable rendering algorithms. Each data representation has unique advantages that make it suitable for particular problems. In the following sections, we will focus on four main representations of 3D models: meshes, voxels, point clouds, and neural implicit representations.

2.2. Mesh

2.2.1 Mesh Rasterization

A mesh represents a 3D shape as a set of vertices and the surfaces that connect them. It is widely used, especially in computer graphics, because it can represent complex 3D shapes in a compact way. In standard rendering, one pixel is assigned with exactly one triangle. The pixel color is determined from the barycentric coordinates of the pixel center projected onto the mesh. Standard rendering leads to bigger problems in calculating gradients in the following four scenarios:

- Pixel color doesn't depend on triangle vertex positions.
- Mesh edge pixels are non-differentiable due to discontinuities.
- Pixels near occlusions lack useful gradients.
- The pixels gradients at vertices are always zero.

The above problems can be solved in two ways: approximated gradients or approximated rendering. On the other hand, if we change our perspective and enforce physical-realistic rendering, we automatically get differentiable rendering.

2.2.2 Approximate Gradient

Loper and Black [20] employ approximated spatial gradients in the first general purpose differentiable renderer, named OpenDR. This method approximates pixel gradients by taking the finite difference from neighboring pixels. This ensures non-zero gradients in all four scenarios described above. We can take one step further. The approximated gradients described above is not object aware. Kato et al [10] developed an object-aware method that approximates gradients using non-local pixels. This method provides more accurate derivatives than OpenDR, as the latter only aims to provide useful derivatives. However, this method is way less efficient in terms of computation.

2.2.3 Approximate Rasterization

Instead of approximating the backward pass, other methods approximate the rasterization of the rendering, in order to encode the relevant parameters. Rhodin et al [31] reinterpret the scene parameters to ensure differentiability. To prevent the discontinuity at sharp edges, they apply a transparency weight that decays from object center towards edges. Liu et al [17] take a similar approach and propose a renderer named Soft Rasterizer. In addition to spacial blurring, it replaces the z-buffer triangle selection with a probabilistic approach, where a triangle has a probability of being projected onto a pixel inversely related to the z distance.

2.2.4 Physical-Realistic Rendering

Physical-realistic rendering offers another perspective to interpret the rendering process other than standard rendering. First, we notice that rendering by nature is an integration operation, where color from different sources are integrated into each pixel. From the signal processing perspective, aliasing artifacts from standard rendering causes discontinuities, where the energy is infinite and gradient is undefined. By applying an anti-aliasing filter, high energies are removed and the resulting pixel gradients are well-defined. Second, even if the integrand of rendering is discontinuous, we can still differentiate the rendering by separating the computation into two components: interior gradients and edge gradients. This can be done on modern light-tracing renderers. [45]

Global illumination is a challenge. Light transport can be arbitrarily complex. There are several difficulties, including

radiance discontinuity at boundary edges, sharp edges and mesh silhouette, and arbitrarily large number of gradients can accumulate on one pixel. To address the first three issues, Li et al and Zhang et al [11] propose methods that separates the computation into two parts: internal gradients and edge gradients, similar to the physical-realistic rendering process described in the previous section. To address the third problem, David et al [26] propose an efficient approach. Instead of storing the computation graph for the forward light transport. They cast light 'backwards' from the camera to the scene during the back propagation. The gradient is calculated from the dot product between the forward light transport against the backward light transport.

2.3. Voxel

In this section we survey differentiable rendering algorithms that use voxels to represent data. Since the position of a voxel is fixed in space, the gradient issue described in the previous section vanishes. The first step of voxel rendering is collecting all voxels along a ray. Here we list 3 approaches:

- Directly collect a subset of voxels in the world space
- Project the voxels to the screen space and perform a bilinear sampling
- Introduce mapping between the template volume and output volume to improce resolution

Since all of the above methods are based on a subset of voxels, the output is differentiable with respect to them

The second step is aggregating the collected voxels to get color. This is done by weighted averaging or weighted sampling the collected voxels. The final color can be further weighted against neighboring voxels

2.4. Point Cloud

Point clouds are collections of data points defined in a three-dimensional coordinate system. These points represent the external surfaces of objects and environments with relatively low storage cost. Each point in a point cloud has specific coordinates (X, Y, Z) and may also include additional attributes like color (RGB values), intensity, normal vectors, or other relevant information. Point clouds can be easily acquired by various 3D sensing technologies, such as RGB-D cameras available on the market nowadays. Due to their simplicity and flexibility, point clouds serve as a fundamental 3D data representation in 3D vision communities and have been integrated into deep learning frameworks for solving practical 3D problems [5, 29].

Rendering a point cloud involves three steps: First, calculate the screen space coordinates p^i_{uv} of each 3D point P^i_{xyz} using matrix multiplication, making this step differentiable. Second, compute each point's influence I_i on the target pixel color. Third, aggregate points based on their influences I and z-values to determine the pixel's final color.

To calculate I_i, a straightforward approach assumes that p^i_{uv} occupies one pixel, but this may result in sparse images. Solutions include increasing the influence area of p^i_{uv}, such as using a truncated Gaussian blur [32, 41], or influence values based on pixel distance [14, 38], making the images differentiable in autodiff frameworks. However, large influence sizes create a trade-off between rendering quality and optimization convergenece by reducing pixel gradient derivatives at distant points. [41] propose an invisible approximated gradient to mitigate this issue.

Several methods exist for computing pixel color. [41] calculates a weighted sum of point colors, but it neglects occlusion. [16] address occlusion by selecting the nearest camera point. [14] suggest selecting the K-nearest 3D points to weigh spatial influence, while [38] add distance-based weighting from the camera. [6] take a different approach by generating 3D voxels from point clouds and rendering them through another method.

Recently, 3D Gaussian Splatting [12] has gained popularity for rendering 3D scenes using Gaussian primitives. Each Gaussian, defined by position, covariance, and SH-coefficients, is projected onto the image plane, blending overlapping regions to create final pixel colors. This method naturally handles occlusion by blending Gaussians based on their spatial influence and provides continuous interpolation, making it more effective than point clouds for achieving high-quality, photo-realistic rendering.

2.5. Implicit Representation

Neural implicit representation [3, 22, 27] represents geometric information with a parametrization of neural networks denoted as F. It takes a 3D point $P^i_{xyz} \in \mathbb{R}^3$ as input, and output with $F(P^i_{xyz})$. Unlike voxel-based methods, implicit representations have constant memory usage with respect to the spatial resolution, enabling querying at infinite resolution without excessive memory consumption during surface reconstruction.

There are three main methods for representing geometric information. First, a neural network F can model the probability that a point P^i_{xyz} is occupied by an object, which becomes a binary classification problem when ground-truth occupancy is provided. Second, F can model transparency at a point, useful for representing semi-transparent and opaque objects. Third, the object's surface can be defined using the level-set method, where the points P^i_{xyz} satisfy $F(P^i_{xyz}) = 0$, indicating the surface boundary, with the sign of F showing whether P^i_{xyz} is inside or outside the surface. Multiple differentiable rendering algorithms have been developed for approximated occupancy probability [39], occupancy probability and transparency [34], and implicit surfaces by distance functions (SDF) [8].

Since obtaining ground truth 3D shapes remains challenging, multiple works leverage differentiable rendering

algorithms [18, 19, 23, 24, 40, 42] and information from 2D images for training. [18] compute occupancy probability along a ray R sampled from a pixel p, whose occupancy probability p_{uv} is assigned by the maximum one along the 3D ray samples p_{xyz}. The gradients are thus backpropagated from p_{uv} to p_{xyz}. NeRF [23] propose rendering neural transparency using volume rendering, allowing gradients to flow across multiple points along a ray for improved optimization. [19, 24, 40, 42] use neural implicit functions with the level set method, where rays are cast to compute camera-to-surface distances. Since the distance to the surface of background pixels is infinite, different loss functions are used for foreground and background pixels with silhouettes.

Point sampling is challenging for neural implicit representations because they do not have a finite set of discrete voxels, as in voxel-based methods. Therefore, many methods employ importance sampling near boundaries [18] and coarse-to-fine sampling along rays [23, 24, 40].

3. Applications

These following applications are discussed in [11]

3.1. Object Reconstruction

Single-view 3D object reconstruction involves estimating the 3D shape of an object from just one image. Unlike methods such as multi-view stereo and structure-from-motion, which rely on multiple images, this task is tackled through machine learning. Recent approaches can generate high-quality 3D reconstructions from natural images. With differentiable rendering, we can set up a single-view reconstruction using self-supervised approach. Thus significantly reduces annotation work.

3.2. Human Reconstruction

Human body and pose reconstruction can open up the possibility for a broad range of real world applications. Currently methods for single view reconstruction are limited. Differentiable rendering can be integrated with machine learning methods that learns prior knowledge to build more accurate shapes

3.3. 9D Reconstruction

It is possible to build a automatic annotation pipeline that recovers 9D pose(translation, rotation, scale) of cars from pretrained off-the-shelf 2D detectors and sparse LiDAR data by incorporating a differentiable renderer.

4. Evaluation

Evaluation of differentiable rendering methods is essential because rendering is a complex function, and accurate evaluation ensures the correctness and applicability of the computed gradients. A direct approach for evaluation involves gradient comparison, such as finite differences for global illumination models [15, 21, 43, 44] and approximated gradients for local illumination models [9, 10, 20]. However, the lack of a common dataset makes it difficult to quantitatively assess the performance of different methods. Many papers have explored other methodologies on evaluating the effectiveness of differentiable rendering algorithms. [10, 15] visualize gradients or assess the convergence efficiency during optimization. [15, 20, 21] report computation time as part of the evaluation. [3, 9, 10, 17, 34, 39] evaluates the image reconstruction accuracy for single-view 3D object reconstruction. [3, 17, 20] directly evaluates optimized scene parameters. Given that large-scale 3D asset datasets [4, 35] have been released, evaluating intrinsic parameters of objects, such as geometry and material properties, could be further facilitated, allowing for more comprehensive evaluation.

5. Libraries

While multiple non-differentiable rendering libraries have been developed over the past few decades, including rasterizer-based methods such as Direct3D [2] and OpenGL [33], as well as ray tracing-based methods like OptiX [28] and Embree [36], this section will instead cover differentiable rendering libraries, including Tensorflow Graphics [1], Kaolin [7], PyTorch3D [30], Mitsuba 3 [25], and Nvdiffrast [13]. A full comparison table of these differentiable renderers is shown in Tab. 1.

5.1. TensorFlow Graphics

TensorFlow Graphics [1] is the first library in this field developed by Google. It aims to provide a set of differentiable graphics layers (e.g., cameras, reflectance models, mesh convolutions) that allows users to easily integrate with TensorFlow-based neural network frameworks, along with 3D viewer functionalities (e.g., 3D TensorBoard) for easier visualization.

5.2. Kaolin

NVIDIA's Kaolin library [7] provides a PyTorch API for working with a range of 3D representations and includes GPU-optimized operations such as differentiable rendering, fast representation conversions, data loading, and a differentiable camera API. It supports 3D object representations like triangle/quad meshes, point clouds, voxel grids, and signed distance functions (SDF). Kaolin also features a model zoo with pretrained weights for various 3D neural architectures and supports differentiable lighting using spherical harmonics and gaussians.

5.3. PyTorch3D

PyTorch3D [30] is being developed by Facebook and is based on the PyTorch deep learning framework. The core

Table 1. **Comparison between existing different differentiable renderers.**

	Tensorflow Graphics [1]	Kaolin [7]	PyTorch3D [30]	Mitsuba 3 [25]	Nvdiffrast [13]
Core implementation	PyTorch / CUDA	PyTorch / CUDA	TensorFlow / C++ / GPU	C++ / CUDA Python bindings	PyTorch Tensorflow
Supported primitives	Triangle mesh Point cloud	Triangle / quad mesh Point cloud Voxel grid SDF	Triangle mesh Point cloud	Triangle mesh Custom	Triangle mesh
DR algorithms	Soft Rasterizer	NMR Soft Rasterizer DIB-Renderer	Analytical derivatives	Loubet *et al.* [21]	CUDA Rasterizer
Rendering Method	Rasterization	Rasterization	Rasterization	Ray tracing	Rasterization
3D operations	Graph convolutions 3D transformations Point Cloud Operations	Primitive conversions 3D transformations	Graph convolutions 3D transformations	3D transformations	3D transformations
Shader support	Hard/soft Phong Hard/soft Gouraud Hard flat Soft silhouette	Phong	Phong	Wide range BSDF	Not built-in
Lighting support	Point Directional	Ambient Directional	Point Spherical harmonic	Area / Point / Spot Constant environment Environment map Directional	Not built-in
Loss functions	Chamfer Distance Mesh edge Laplacian smoothing Normal consistency	Chamfer distance Directed distance Mesh Laplacian	Chamfer distance	Not supported	Not supported
Camera support	Perspective Orthographic	Perspective Orthographic	Perspective Orthographic Quadratic distortion	Perspective Irradiance meter Radiance meter	Not built-in
Data I/O support	OBJ / PLY	External	External	OBJ / PLY	External

rendering algorithm and all of its 3D operators are optimized through CUDA implementations for both the forward and backward passes. Multiple 3D loss functions, such as Chamfer distance [37], Laplacian regularization loss [37], and normal consistency loss [37], are supported in PyTorch3D. PyTorch3D also supports heterogeneous batching of 3D data, allowing the renderer to input meshes of different sizes.

5.4. Mitsuba 3

Mitsuba 3 [25] is a research-oriented high-performance rendering system for forward and inverse light-transport simulation. It consists of a small set of core libraries and a wide variety of plugins that implement functionality ranging from materials and light sources to complete rendering algorithms. For optimization purposes, its Just-in-time (JIT) compiler supports symbolic execution of algorithms on GPU. Auto-differentiation is also supported through the Enoki library.

5.5. Nvdiffrast

Nvdiffrast [13] is a PyTorch/TensorFlow library developed by NVIDIA that provides high-performance primitive operations for rasterization-based differentiable rendering. These primitives allow users to build custom, high-performance graphics pipelines directly within automatic differentiation frameworks. The library is designed for modularity and flexibility, enabling integration into various inverse rendering applications.

6. Summary

This survey examines the current state of Differentiable Rendering (DR), focusing on algorithms, evaluation methods, applications, and available libraries in the field. It demonstrates how DR enables end-to-end optimization of 3D scene parameters using 2D image data. The survey first reviews DR algorithms across various 3D representations—meshes, voxels, point clouds, and implicit representations—highlighting the challenges and solutions for each. It also explores evaluation methods, applications in object and human reconstruction, and provides a comparison of existing DR libraries. This paper offers a comprehensive overview of the research landscape in differentiable rendering.

References

[1] Paige Bailey, Sofien Bouaziz, Shan Carter, Josh Gordon, Christian Häne, Alexander Mordvintsev, Julien Valentin, and Martin Wicke. Differentiable graphics with tensorflow 2.0. *ACM SIGGRAPH 2019 Courses*, pages 1–211, 2019. 4, 5

[2] David Blythe. The direct3d 10 system. *ACM SIGGRAPH 2006 Papers*, pages 724–734, 2006. 4

[3] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5939–5948, 2019. 3, 4

[4] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, et al. Objaverse-xl: A universe of 10m+ 3d objects. *arXiv preprint arXiv:2307.05663*, 2023. 4

[5] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan D Wegner, Konrad Schindler, and Marc Pollefeys. Semantic3d. net: A new large-scale point cloud classification benchmark. *arXiv preprint arXiv:1704.03847*, 2017. 3

[6] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. *Advances in neural information processing systems*, 31, 2018. 3

[7] Krishna Murthy Jatavallabhula, Edward Smith, Jean-Francois Lafleche, Clement Fuji Tsang, Artem Rozantsev, Wenzheng Chen, Tommy Xiang, Rev Lebaredian, and Sanja Fidler. Kaolin: A pytorch library for accelerating 3d deep learning research. *arXiv preprint arXiv:1911.05063*, 2019. 4, 5

[8] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1251–1261, 2020. 3

[9] Hiroharu Kato and Tatsuya Harada. Learning view priors for single-view 3d reconstruction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9778–9787, 2019. 4

[10] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018. 2, 4

[11] Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*, 2020. 3, 4

[12] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023. 3

[13] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (ToG)*, 39(6):1–14, 2020. 4, 5

[14] Lei Li, Siyu Zhu, Hongbo Fu, Ping Tan, and Chiew-Lan Tai. End-to-end learning local multi-view descriptors for 3d point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1919–1928, 2020. 3

[15] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018. 4

[16] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. Learning efficient point cloud generation for dense 3d object reconstruction. In *proceedings of the AAAI Conference on Artificial Intelligence*, 2018. 3

[17] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7708–7717, 2019. 2, 4

[18] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. Learning to infer implicit surfaces without 3d supervision. *Advances in Neural Information Processing Systems*, 32, 2019. 4

[19] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2019–2028, 2020. 4

[20] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VII 13*, pages 154–169. Springer, 2014. 2, 4

[21] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019. 4, 5

[22] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019. 3

[23] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for

view synthesis. *Communications of the ACM*, 65(1): 99–106, 2021. 4

[24] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3504–3515, 2020. 4

[25] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics (ToG)*, 38(6):1–17, 2019. 4, 5

[26] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. Radiative backpropagation: An adjoint method for lightning-fast differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39 (4):146–1, 2020. 3

[27] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019. 3

[28] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. Optix: a general purpose ray tracing engine. *Acm transactions on graphics (tog)*, 29(4):1–13, 2010. 4

[29] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 3

[30] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020. 4, 5

[31] Helge Rhodin, Nadia Robertini, Christian Richardt, Hans-Peter Seidel, and Christian Theobalt. A versatile scene model with differentiable visibility applied to generative pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 765–773, 2015. 2

[32] Riccardo Roveri, A Cengiz Öztireli, Ioana Pandele, and Markus Gross. Pointpronets: Consolidation of point clouds with convolutional neural networks. In *Computer Graphics Forum*, pages 87–99. Wiley Online Library, 2018. 3

[33] Dave Shreiner et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education, 2009. 4

[34] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2626–2634, 2017. 3, 4

[35] Benjamin Ummenhofer, Sanskar Agrawal, Rene Sepulveda, Yixing Lao, Kai Zhang, Tianhang Cheng, Stephan Richter, Shenlong Wang, and German Ros. Objects with lighting: A real-world dataset for evaluating reconstruction and rendering for object relighting. In *2024 International Conference on 3D Vision (3DV)*, pages 137–147. IEEE, 2024. 4

[36] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. Embree: a kernel framework for efficient cpu ray tracing. *ACM Transactions on Graphics (TOG)*, 33(4):1–8, 2014. 4

[37] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–67, 2018. 5

[38] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7467–7477, 2020. 3

[39] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. *Advances in neural information processing systems*, 29, 2016. 3, 4

[40] Lior Yariv, Matan Atzmon, and Yaron Lipman. Universal differentiable renderer for implicit neural representations. *arXiv preprint arXiv:2003.09852*, 2, 2020. 4

[41] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019. 3

[42] Sergey Zakharov, Wadim Kehl, Arjun Bhargava, and Adrien Gaidon. Autolabeling 3d objects with differentiable rendering of sdf shape priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12224–12233, 2020. 4

[43] Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. A differential theory of radiative transfer. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019. 4

[44] Cheng Zhang, Zihan Yu, and Shuang Zhao. Path-space differentiable rendering of participating media.

ACM Transactions on Graphics (TOG), 40(4):1–15, 2021. 4

[45] Shuang Zhao, Wenzel Jakob, and Tzu-Mao Li. Physics-based differentiable rendering: from theory to implementation. *ACM siggraph 2020 courses*, pages 1–30, 2020. 2