Differentiable Rendering: A Survey

Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka,
Wadim Kehl and Adrien Gaidon

Abstract— Deep neural networks (DNNs) have shown remarkable performance improvements on vision-related tasks such as object detection or image segmentation. Despite their success, they generally lack the understanding of 3D objects which form the image, as it is not always possible to collect 3D information about the scene or to easily annotate it. Differentiable rendering is a novel field which allows the gradients of 3D objects to be calculated and propagated through images. It also reduces the requirement of 3D data collection and annotation, while enabling higher success rate in various applications. This paper reviews existing literature and discusses the current state of differentiable rendering, its applications and open research problems.

Index Terms—Differentiable Rendering, Inverse Graphics, Analysis-by-Synthesis

1 Introduction

The last years have clearly shown that neural networks are effective for 2D and 3D reasoning [1], [2], [3], [4], [5], [6]. However, most 3D estimation methods rely on supervised training regimes and costly annotations, which makes the collection of all properties of 3D observations challenging. Hence, there have been recent efforts towards leveraging easier-to-obtain 2D information and differing levels of supervision for 3D scene understanding. One of the approaches is integrating graphical rendering processes into neural network pipelines. This allows transforming and incorporating 3D estimates into 2D image level evidence.

Rendering in computer graphics is the process of generating images of 3D scenes defined by geometry, materials, scene lights and camera properties. Rendering is a complex process and its differentiation is not uniquely defined, which prevents straightforward integration into neural networks.

Differentiable rendering (DR) constitutes a family of techniques that tackle such an integration for end-to-end optimization by obtaining useful gradients of the rendering process. By differentiating the rendering, DR bridges the gap between 2D and 3D processing methods, allowing neural networks to optimize 3D entities while operating on 2D projections. As shown in Figure 1, optimization of the 3D scene parameters can be achieved by backpropagating the gradients with respect to the rendering output. The common 3D self-supervision pipeline is applied by integrating the rendering layer to the predicted scene parameters and applying the loss by comparing the rendered and input image in various ways. The applications of this process are broad, including image-based training of 3D object reconstruction [7], [8], [9], human pose estimation [10], [11], hand pose estimation [12], [13] and face reconstruction [14], [15].

- *Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando and Toru Matsuoka are with Preferred Networks, Inc. E-mail: {hkato, dbeker, mmorariu, ando, tmatsuoka}@preferred.jp*
- *Wadim Kehl is with Toyota Research Institute - Advanced Development. E-mail: wadim.kehl@tri-ad.global*
- *Adrien Gaidon is with Toyota Research Institute. E-mail: adrien.gaidon@tri.global*

Despite its potential, using existing or developing new DR methods is not straightforward. This can be attributed to four reasons:

- Many DR-based methods have been published over the past few years. To understand which ones are suitable for addressing certain types of problems, a thorough understanding of the methods' mechanisms as well as the underlying properties is required.
- In order to choose or develop a novel DR method, the evaluation methodology of the existing methods should be known.
- In order to use DR in a novel task, it is necessary to survey the usage of DR in existing applications. However, due to the variety of applications, it is not trivial to have a clear view of the field.
- Several DR libraries have emerged over the past years with each focusing on different aspects of the differentiable rendering process. This makes some of the libraries useful for a particular type of applications, while for others, additional functionality may need to be implemented from scratch. Also, certain applications are constrained by computational requirements that existing implementations of DR-based methods often do not fulfill. This is especially the case for real-time applications or embedded devices, for which highly optimized neural networks are necessary.

To address these shortcomings, the current state of DR needs to be properly surveyed. However, to the best of our knowledge, there has been no comprehensive review for this purpose to date.

In this work, we provide an overview of the current state of DR algorithms in Section 2, the evaluation metrics in Section 3, the applications that use differentiable rendering in Section 4 and the libraries that are currently being used to facilitate the research in Section 5. Besides offering a survey on the existing methods, we also discuss open research problems and provide suggestions for future work.

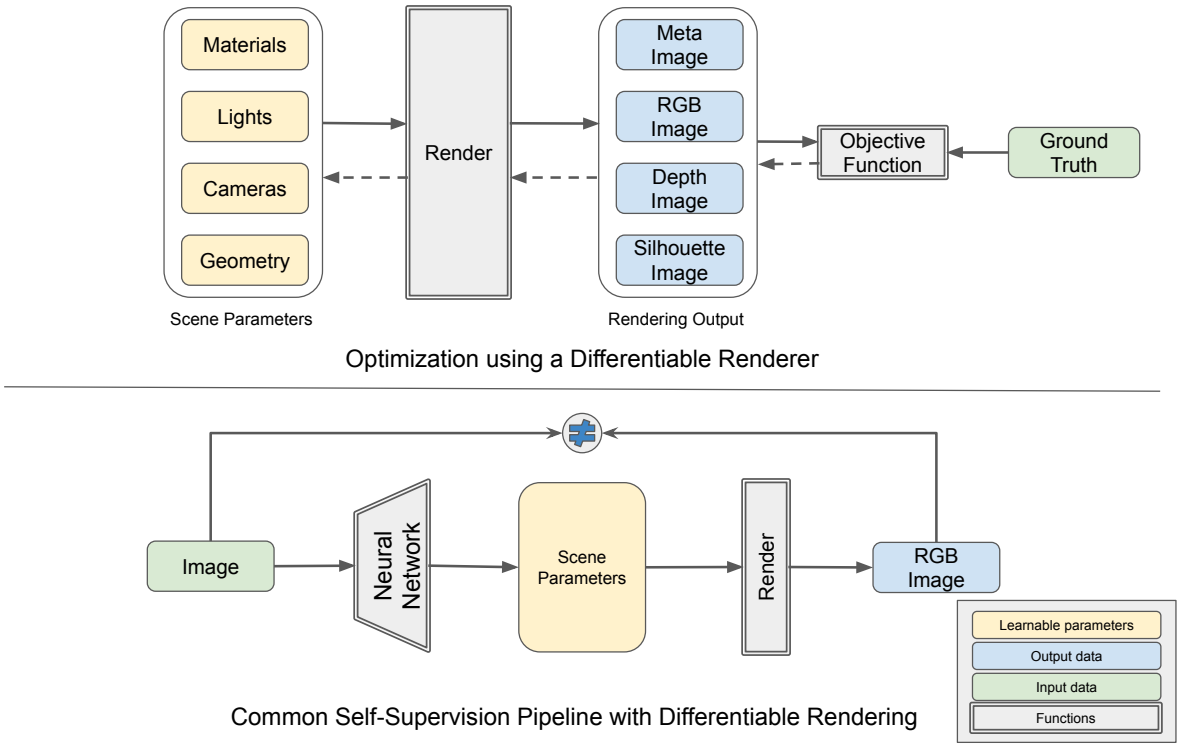

Fig. 1. Schematic overview of differentiable rendering. Best viewed in color. The top part shows a basic optimization pipeline using a differentiable renderer where the gradients of an objective function with respect to the scene parameters and known ground-truth are calculated. The bottom part shows a common self-supervision pipeline based on differentiable rendering. Here, the supervision signal is provided in the form of image evidence and the neural network is updated by backpropagating the error between the image and the rendering output.

2 Algorithms

TABLE 1
Overview of the representative differentiable rendering methods. They are classified by the four main underlying data representations.

Data Repr	Type	Literature
Mesh	Analytical derivative	[16], [17], [18]
	Approx. gradient	[19], [9], [20], [14]
	Approx. rendering	[21], [22], [23]
	Global illumination	[24], [25], [26], [27], [28]
Voxel	Occupancy	[7], [29], [30], [31]
	SDF	[32]
Point cloud	Point cloud	[33], [34], [35], [36], [37], [38]
	RGBD image	[39], [40]
Implicit	Occupancy	[41], [42]
	Level set	[43], [44], [45], [46]

We start by briefly defining a mathematical formulation for our purposes. A rendering function $\mathcal{R}$ takes shape parameters Φ_s, camera parameters Φ_c, material parameters Φ_m and lighting parameters Φ_l as input and outputs an RGB image I_c or a depth image I_d. We denote the inputs as $\Phi = \{\Phi_s, \Phi_m, \Phi_c, \Phi_l\}$ and the outputs as $I = \{I_c, I_d\}$. Note that a general DR formulation can have different kinds of additional input/output entities, but in this section we refer to the most common ones. A differentiable renderer computes the gradients of the output image with respect to the input parameters $\partial I/\partial \Phi$ in order to optimize for a specific goal (loss function). The computation of these gradients can be approximate, but should be accurate enough to propagate meaningful information required to minimize the objective function.

The inputs to a rendering function Φ, especially the geometric parameters Φ_s, are some of the main differentiators between the differentiable rendering algorithms. Each data representation has its own strengths which makes it suitable for solving specific problems. In this work we group the surveyed DR algorithms into four categories, based on the underlying data representation: mesh, voxels, point clouds and neural implicit representations. We also discuss *neural rendering*, as there is a growing body of research into learning to render using neural network models, rather than designing the rendering and its differentiation manually. Table 1 lists the literature that we are covering in this section.

2.1 Mesh

2.1.1 Analytical Derivative of Rendering

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.

Given the rendering inputs and a pixel, the process of determining its color can be divided into (1) assigning a triangle to the pixel and (2) computing the color of the pixel based on the colors of the assigned triangle's vertices. The former is done by projecting all mesh triangles from world space to screen space and determining the ones that

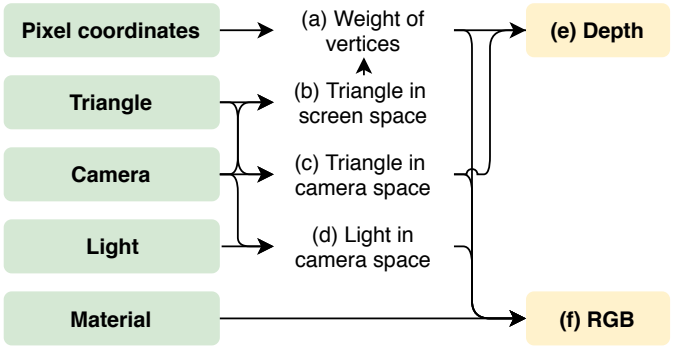

Fig. 2. Several operations that are performed inside a rendering function, given a pixel, its corresponding triangle and material defined on vertices of the triangle, camera parameters, and light configurations. The green boxes represent inputs and the yellow boxes represent outputs. Best viewed in color.

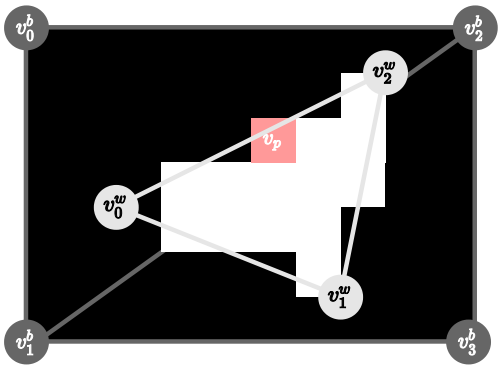

Fig. 3. An image of 10×7 pixels that shows a scene composed of three triangles. The vertex colors of one are white and its vertex positions are denoted by v_i^w. The vertex colors of the other two are black and their vertex positions are denoted by v_i^b.

enclose the pixel. The triangle that is closest to the camera is then selected. Since the selection yields a discrete triangle identifier, this operation is not differentiable with respect to all parameters.

All further operations are differentiable. Figure 2 illustrates the simplified computation flow. First, the triangle, the position and direction of the lights are projected from world space to camera space (Figure 2 (c), (d)), followed by a transformation into screen space (Figure 2 (b)). These operations are differentiable because they are accomplished via simple matrix products. The pixel coordinates can then be expressed as a weighted sum of three vertices. The computation of the weights (Figure 2 (a)) is done by solving (differentiable) linear programs whereas the pixel depth is computed by interpolating the depth of the three vertices in camera space (Figure 2 (d)). Similarly, the material and normal vectors at a pixel are typically expressed as a weighted sum of the material and normal vectors defined at the triangle's vertices. For local illumination models, given the material and lighting parameters, as well as the normal vector at a pixel, the pixel color can be computed using a reflection model (Figure 2 (e)). Popular reflection models such as Phong [47], Lambertian [48] and Spherical Harmonics [49] are all differentiable. Therefore, the derivatives of the pixel depth and color with respect to the input parameters can be computed analytically [16], [17], [18].

In standard rendering, exactly one triangle per pixel is typically selected to compute the final color value, which can lead to optimization problems. To exemplify, let us consider a scene composed of one white triangle and two black triangles as illustrated in Figure 3. The vertex color of v_i^w is 1 and the vertex color of v_i^b is 0. Then, using barycentric coordinates w_i that satisfy $v_p = \sum w_i v_i^w$ and $\sum w_i = 1$, the color of a pixel at v_p is a constant $c_{v_p} = \sum w_i c_{v_i^w} = 1$. Therefore, the gradient of c_{v_p} with respect to v_i^w and v_i^b is zero.

Similarly, the derivative of a pixel color with respect to a vertex position is always zero for all pixels and vertices. Therefore, analytical derivatives do not help with optimizing the geometry in this case. However, in practice, the position of vertices affects pixel colors. For example, when v_2^w moves to the right, c_{v_p} would change to 0. We can therefore solve the problem by allowing the color of unrelated pixels to affect neighboring triangles. Several rendering methods provide approximated gradients that reflect these insights [19], [9], [20], [14], while others overcome this problem by approximating the rasterizer pass [22], [23], [21].

2.1.2 Approximated Gradients

Loper and Black [19] employ approximated spatial gradients in the first general purpose differentiable renderer, named *OpenDR*. v_p can be represented by $v_p = \sum w_i v_i^w$ with $\sum w_i = 1$ and the pixel derivatives with respect to v_p can be computed using differential filters (e.g. Sobel filter). In other words, $\{\frac{\partial c_{v_p}}{\partial x}, \frac{\partial c_{v_p}}{\partial y}\} = \frac{\partial c_{v_p}}{\partial v_p}$. Because the pixels that are located to the left and right of v_p are taken into consideration when computing the gradient, it can have a non-zero value in this formulation.

Kato *et al*. [9] raise two issues with OpenDR and propose a renderer named *neural 3D mesh renderer (NMR)*. The first issue is the localness of gradient computation. Because of the localness of differential filters in OpenDR, only gradients on boundary pixels can flow towards vertices, whereas gradients at other pixels cannot be used. Optimization based on this property may lead to poor local minima. The second issue is the derivative does not leverage the loss gradient of the target application, e.g. image reconstruction. For example, in the case of Figure 3, if the objective is to decrease the intensity of v_p, we should displace v_2^w. However, if the objective is to increase it, we should not. Therefore, gradients should be objective-aware for better optimization. Since the objective of OpenDR is not to provide accurate gradients but to provide useful gradients for optimization, a loss-aware gradient flow is required for this purpose. To overcome these issues, the authors propose non-local approximated gradients that also use gradients of pixels backpropagated from a loss function. The authors later replaced non-local gradients with local gradients similar to OpenDR, in order to reduce the computation complexity [20].

Genova *et al*. [14] calculate the rasterization derivatives using the barycentric coordinates of each triangle with respect to each pixel. They introduce negative values for the barycentric coordinates of the pixels that lie outside the triangle border, in order to overcome the occlusion discontinuity. By omitting triangle identifiers and by employing

negative barycentric coordinates, shapes can be treated as being locally planar, to approximate the occlusion boundaries. However, such approximation could pose problems when optimizing for translation or occlusion.

2.1.3 Approximated Rendering

Instead of approximating the backward pass, other methods approximate the rasterization (or the forward pass) of the rendering, in order to be able to compute useful gradients.

Rhodin *et al*. [21] reinterpret the scene parameters to ensure differentiability. To prevent the discontinuity at hard object boundaries, each object is defined by a density parameter which has the maximum opaqueness at the object's center and becomes transparent towards the boundaries. As a result, the rendering result is blurry and smooth towards the edges, while removing sharp corners from the scene parameters ensures differentiability.

Liu *et al*. [22] take a similar approach and propose a renderer named *Soft Rasterizer*. In addition to spatial blurring, it replaces the z-buffer-based triangle selection of a vanilla rasterization process with a probabilistic approach in which each triangle that is projected onto a pixel p_i contributes to its color with a certain probability. In practice, an aggregation function fuses all the color probabilities for each pixel. As a result, each pixel color is computed as a weighted sum of the values corresponding to the relevant triangles and that operation is differentiable. The weights are based on the distances between a pixel and a triangle in the 2D screen space, as well as the distance between the camera and the triangle, along the viewing direction. Therefore, gradients accumulate information across the whole image in a probabilistic way, while OpenDR only backpropagates to a vertex from neighbouring pixels and NMR only backpropagates gradients for the pixels that lie within $[min(obj_x), max(obj_x)]$ and $[min(obj_y), max(obj_y)]$. Note that all methods in the previous section provide no control over the forward pass as they aim to approximate the backward gradients only.

Chen *et al*. [23] propose *DIB-R*, which focuses independently on two different regions of the image: foreground pixels, where a pixel is covered by at least one face, and background pixels, which do not have any face coverage. To avoid the blurry output of Soft Rasterizer, DIB-R proposes to use analytical derivatives for foreground pixels, computed using the barycentric interpolation of a face's vertex attributes. It also prevents vanishing gradients for background pixels by employing a distance-based aggregation of global face information in a similar way to Soft Rasterizer.

2.1.4 Global Illumination

The pipeline in Figure 2 does not hold for global illumination models because lighting for a pixel is not affected by reflected light from other surface points. Although this simplification reduces the rendering time, the generation of photorealistic images that contain complex interactions of light, geometry, and materials becomes impossible. The color of a pixel is computed using the Monte Carlo estimation of the rendering equation [50] in global illumination models. The main challenge in differentiable photorealistic rendering is estimating the derivative corresponding to the integral of the Monte Carlo-estimated rendering equation when the integral contains discontinuities due to object silhouettes.

Li *et al*. [24] is the first work to compute derivatives of scalar functions over a physically-based rendered image with respect to arbitrary input parameters like camera, light materials and geometry. It uses a stochastic approach based on Monte Carlo ray tracing which estimates both the integral and the gradient of the pixel filter's integral. As edges and occlusions are discontinuous by nature, the integral calculation is split into smooth and discontinuous regions. For the smooth parts of the integrand, a traditional area sampling with automatic differentiation is employed. For the discontinuous parts, a novel edge sampling method is introduced to capture the changes at boundaries. Their method makes certain assumptions: meshes do not have interpenetration, there are no point light sources, no perfectly specular surfaces and the scene is static. Zhang *et al*. [25] propose a very similar method. Different from Li *et al*. [24], their approach supports the differentiation of volumes in addition to triangle meshes. Two major drawbacks of these methods are the rendering speed and the large variance of the estimated gradients. This is due to the challenging task of finding all object edges and sampling them, for which many samples are required.

Instead of relying on edge sampling, Loubet *et al*. [26] propose to reparametrize all relevant integrals, including pixel integrals over spherical domains. Discontinuity occurs at those points that depend on a scene parameter, therefore they reparametrize the variables from the integrals to remove this dependency. Such reparametrizations allow integration over a space where discontinuity does not take place when a scene parameter changes and is equivalent to importance sampling the integral that follows the discontinuity. Even though this approach is computationally efficient, it does not support perfectly specular materials, degenerate light sources containing Dirac delta functions and multiple discontinuities within the support of the integrand. In addition, since the gradients are approximated, they may not always be accurate.

Zhang *et al*. [27] propose a method to estimate the derivatives of the path integral formulation [51] while all the methods reviewed above address the discontinuity problem in the rendering equation [50]. The authors show that the differentiation of path integrals can be separated into *interior* and *boundary* terms and propose Monte Carlo methods for estimating both components. The proposed method is unbiased and computationally efficient as it does not need to find object silhouette edges explicitly. However, because the computation of the gradients for a single rendered image takes anywhere between a few second to tens of seconds, it is impractical for training a neural network.

The gradient with respect to material and light parameters can be computed by automatic differentiation. However, given its large memory footprint, applications are limited to simple scenes. To address this issue, Nimier-David *et al*.[28] propose an efficient approach to gradient computation. In their method, a computational graph is not stored during rendering. Instead, during backpropagation, rays with gradients are cast from the camera and gradients are propagated to surfaces at occlusion. However, optimizing shapes using this method is challenging because

a change in object visibility is not differentiable and not considered during backpropagation.

2.2 Voxel

In this section we survey differentiable rendering algorithms that use voxels to represent data. A voxel is a unit cube representation of a 3D space. It can be parametrized as an N-dimensional vector that contains information about the volume that is occupied in 3D space, as well as additional information. It is common practice to encode occupancy information in a voxel using a binary value or transparency using a non-binary one. For applications where occupancy is predicted, non-binary occupancy probability $P_o \in [p_{min}, p_{max}]$ is usually stored. Even though occupancy probabilities are different from transparency values, they can be interpreted the same way in order to maintain differentiability during the ray marching operation. In this case, the probability P_o denotes a ray's absorption (transparency) at a certain point. Material information is often also stored. Instead of storing occupancy, shapes can be represented as the shortest distance from the center of each voxel to the object's surface. In this representation, each voxel cell is assigned a distance function (DF). Distance functions can be augmented with a signed value denoting whether the voxel is contained inside or outside the object, to form a Signed Distance Function (SDF). Alternatively, truncation can be applied to an SDF to form a Truncated Signed Distance Function (TSDF) for those applications where only the distance information to the object's surface is important.

All the voxels that are located along a ray that projects to a pixel are taken into account when rendering that pixel. Several approaches exist for deciding the resulting pixel color [7], [29], [30], [31], [32]. Since the position of a voxel is fixed in the 3D space, the gradient issue described in Section 2.1, which is caused by the displacement of shape primitives, does not occur when rendering voxels.

Collecting the voxels that are located along a ray is the first step of the rendering process. Tulsiani *et al*. [29] and Jiang *et al*. [32] perform this operation in world space, while Yan *et al*. [7] and Henzler *et al*. [30] take a different approach. They project the voxels from the world space to the screen space (using camera parameters) and perform a bilinear sampling similar to Spatial Transformer [52], which is more computationally efficient. The approach of Lombardi *et al*. [31] is rather different. They introduce the notion of warp field (which is the mapping between the template volume and output volume) to improve the apparent resolution and reduce grid-like artifacts along with jagged motion. The inverse wrap value is evaluated and the template volume is sampled at the warped point. Since all the described operations are based on a subset of voxels, the output is differentiable with respect to them.

Aggregating voxels along a ray is the second step of the rendering process. Yan *et al*. [7] associate an occupancy probability to each pixel. This value is computed by casting a ray from the pixel, sampling all the associated voxels and choosing the one with the highest occupancy probability. Instead of taking the maximum value, Tulsiani *et al*. [29] compute the probability that a ray stops at a certain distance. The advantage of this method is the ability to render depth maps and color images in addition to the foreground mask of an object. Henzler *et al*. [30] further introduce the bidirectional emission-absorption (EA) light radiation model for the emitting material, in addition to the visual hull (VH) and absorption-only (AO) models which are similar to Tulsiani *et al*.. Lombardi *et al*. [31] employ differentiable ray marching by casting a ray from each pixel and then iteratively accumulating color and opacity probabilities along it. When the cumulative sum of the opacity values along the ray reaches a maximum value, the remaining voxels are discarded in order to prevent further impact to its color.

Different from these aforementioned methods, Jiang *et al*. [32] handle voxels that represent signed distance functions. They cast a ray from the pixel and locate the surface voxel that is closest to the camera, along the ray. The final pixel color is computed as a weighted sum of the surface voxel and its neighbouring voxels. They also compute a normal vector at the surface point, based on the voxel values around it, in order to compute a shading value. Although locating the surface point is not a differentiable operation, the color of the pixel is differentiable with respect to the voxels around the surface point.

Several works also take materials into account during the rendering process. Tulsiani *et al*. [29], Henzler *et al*. [30], and Lombardi *et al*. [31] simply use voxel colors to represent the material. Similar to mesh rendering, the rendered image is differentiable with respect to the material parameters, since most of the shading models are differentiable.

2.3 Point Cloud

Point clouds are collections of points that represent shapes in the 3D space. They are ubiquitous in the 3D vision community due to their ability to represent a large variety of topologies at a relatively low storage cost. Moreover, most of the 3D sensors that are nowadays available on the market also rely on point clouds for encoding data. Recent years have shown that point clouds can successfully be integrated into deep neural networks to solve a variety of practical 3D problems [6], [53], [5]. Given the advent of differentiable rendering and its potential for scene understanding with reduced 3D supervision, point clouds have thus become a natural choice for data representation.

Rendering a point cloud is done in three steps. First, the screen space coordinates p^i_{uv} of a 3D point P^i_{xyz} is calculated. This operation is achieved by a matrix multiplication as with rendering of mesh. Therefore, this step is differentiable. Second, the influence I_i of each 3D point on the color of target pixel's color is computed. Third, the points are aggregated based on their influences and z-values, in order to decide the final color value for a pixel. Several methods for addressing this step have been proposed as well.

The straightforward way to calculate the I_i is to assume that p^i_{uv} has a size of one pixel [36]. However, this approach may lead to very sparse images. One way to solve this problem is creating influences larger than one pixel per p^i_{uv} in the image. Several papers employ a truncated Gaussian blur [33], [34] or influence values based on the distance between a pixel and p^i_{uv} [35], [37]. The generated images are differentiable with respect to the points when these operations are implemented in an autodiff framework.

However, in these methods, there is a trade-off between the rendering quality and the optimization's convergence. A large influence size prevents the derivative of a pixel with respect to a distant P^i_{xyz} from becoming zero, but reduces the rendering quality. To address this issue, Yifan *et al*. [34] propose the invisible approximated gradient, similar to Kato *et al*. [9].

Several methods have been proposed for computing the color of a pixel. A straightforward way is computing a weighted sum of the points' colors, based on the points' influences [34]. However, this approach does take occlusion into account. Lin *et al*. [36] address this issue by selecting the closest point to the camera for a given pixel, which prevents the optimization of the occluded points. Instead, Li *et al*. [37] propose to select the K nearest 3D points P_{xyz} to the camera and weigh them based on the spatial influence I^i at each pixel. Wiles *et al*. [35] propose to weigh all the 3D points P_{xyz} according to the distances from the camera to P_{xyz}, in addition to weighing by spatial influence I^i, similar to Li *et al*. [22].

Insafutdinov *et al*. [38] take an unique approach. Instead of computing influence values and aggregating points in screen space, they generate 3D voxels from the point cloud and render them using a method by Tulsiani *et al*. [8].

A set of depth image and camera parameters is an alternative way to represent point cloud data. One of the major benefits of using a depth image is that its spatial information can be used directly to recover information about the triangulation between points. On the other hand, when projecting a depth image to the 3D space, the point density decreases inversely proportional to the distance. Generation of another view from an depth image can be regarded as a special case of rendering of point clouds. In self-supervised learning of monocular depth estimation [39], [40], a color image is used to estimate the corresponding depth, which is converted to a point cloud and projected to a virtual camera view.

2.4 Implicit Representations

Recently, there has been a growing interest in representing geometric information in a parametrized manner in neural networks [54], [55], [56]. This is commonly known as *neural implicit representation*. In this model, the geometric information at a point $P^i_{xyz} \in \mathbb{R}^3$ is described by the output of a neural network $F(P^i_{xyz})$. Unlike voxel-based methods, in implicit representations the memory usage is constant with respect to the spatial resolution. Therefore, a surface can be reconstructed at infinite resolution without excessive memory footprint.

Similar to voxel-based methods, there are three ways to represent the geometric information. First, the probability that a point P^i_{xyz} is occupied by an object can be modeled by a neural network F. When the ground-truth occupancy at P^i_{xyz} is given, learning F becomes a binary classification problem and has been extensively studied in the literature. Second, F can model the transparency at a point p. This approach can be used for representing semi-transparent objects, as well as approximating the probability of occupancy for scenes with no semi-transparent objects. Third, the surface of an object can be defined as the set of points P^i_{xyz} which satisfy $F(P^i_{xyz}) = 0$ and which is called *level-set method*. Typically, $F(P^i_{xyz})$ defines the distance to the boundary of the target surface, while its sign indicates whether P^i_{xyz} is inside or outside the surface.

As with other representations, obtaining ground-truth 3D shapes is often expensive or impossible, for real-world scenarios. To address this issue, several works propose to use 2D supervision (in the form of depth maps or multi-view images) and leverage the power of differentiable rendering [41], [42], [43], [44], [45], [46]. Similar to voxel-based methods, for which various differentiable rendering algorithms have been developed for approximated occupancy probability [7], occupancy probability and transparency [29] and implicit surfaces by distance functions [32], implicit representations also require implementations to handle the different input types. Table 2 summarizes the similarities between methods based on voxels and the ones based on neural implicit functions.

Liu *et al*. [41] propose the first usage of neural implicit representations in differentiable rendering. To find the occupancy probability of a pixel p_{uv}, a ray R is cast from p_{uv}. Then, the 3D point p_{xyz} with the maximum occupancy probability value along the ray R is selected. Finally, the occupancy probability of p_{uv} is assigned the value of p_{xyz}. The gradient value of p_{uv} is assigned to p_{xyz} during backpropagation. By contrast, Mildenhall *et al*. [42] propose differentiable rendering for neural transparency. In their paper, a pixel value is computed using volume rendering and by weighing the values of all the sampled points along a ray. Therefore, gradients flow into more than one point, which is expected to stabilize optimization. Although Liu *et al*. only support the rendering of silhouettes, Mildenhall *et al*. support the rendering of RGB images using Texture Fields [57]. Another important aspect regarding the work by Mildenhall *et al*. is that they take the direction of a ray into account when computing color values and, therefore, support view-dependent phenomena such as specular reflection.

Several works [43], [44], [45], [46] that employ neural implicit functions represent surfaces by the level set method in which a ray is cast from each pixel and the intersection between the ray and the surface is used to compute the camera-to-surface distance. The color at the intersection point is sampled from a neural network. The derivative of the distance with respect to the intersection point cannot be computed by an autodiff framework, but it can be computed analytically. These methods cannot render differentiable silhouette images because the distance to the surface is infinite for background pixels. Therefore, to optimize while taking silhouettes into account, it is common to employ different loss functions for foreground and background pixels, respectively. For example, Niemeyer *et al*. [43] minimize the occupancy of the intersection point when encountering the projection to the silhouette is a false positive and maximize the occupancy of a random point along a ray when the projection is a false negative.

Sampling points on a ray is easy for voxel-based methods, as there is a finite set of voxels along it, due to the discrete nature of the 3D grid. For neural implicit representations, there is an infinite set of points along the ray and the sampling process becomes challenging. Two of the most

TABLE 2
Comparison of differentiable rendering methods for voxels and neural implicit functions.

Operation	Approach	Voxel	Neural implicit functions
Data collection along a ray	Sampling along a ray	[29], [31], [32]	[42], [43], [44], [45], [46]
	(Re)sampling in 3D space	[7], [30]	[41]
Data aggregation along a ray	Approx. occupancy: taking the maximum value	[7]	[41]
	Transparency: taking a weighted sum	[29], [30], [31]	[42]
	Level set: taking the value at a hit point	[32]	[43], [44], [45], [46]

common ways to address this issue are sampling random points or regular points with random perturbations [42], [43], [44], [45], [46]. Another approach is sampling random points in the 3D space and checking their intersection with the rays, which becomes even more efficient when multiple views are rendered [41]. To optimize even further, many methods employ coarse-to-fine sampling on rays [42], [43], [44] and importance sampling near boundaries [41].

2.5 Neural Rendering

Instead of handcrafting the rendering differentiation, Eslami *et al*. [58] propose to learn the rendering process from data. This approach is commonly known as *neural rendering*. Typically, a scene generation network that outputs a neural scene representation and a rendering network are jointly trained by minimizing the image reconstruction error. Thanks to the recent advances in neural networks, neural rendering is nowadays becoming capable of generating high-quality images and is used for many applications such as novel view synthesis, semantic photo manipulation, facial and body reenactment, relighting, free-viewpoint video and to the creation of photo-realistic avatars.

While handcrafted renderers do not always model the physical world accurately, by learning from real-world data, neural rendering can produce novel images that are indistinguishable from the real-world ones. On the other hand, generalizing to scenes that differ from the training data, scaling to scenes composed of multiple objects and the ability to be modified by humans are limited. One promising direction for improving neural rendering is adding inductive biases for 3D scenes and rendering into neural networks [59], [31], [60]. Therefore, combining differentiable renderers which are based on inductive biases with neural rendering would be one interesting research area to explore.

Please refer to a recent survey [61] for more details about neural rendering.

2.6 Summary

We have presented differentiable rendering techniques grouped based on four data representation types: mesh, voxel, point cloud and implicit functions. Figure 4 illustrates these representations. The main points of this section can be summarized as follows:

Mesh-based approaches can be grouped into three categories: approximated gradients, approximated rendering and global illumination. Gradient approximation allows for efficient and high-quality rasterization while aiming to handcraft meaningful gradients. Rendering approximation may produce blurry output, but ensures non-zero gradients for all pixels. Global illumination-based techniques reduce the domain gap between the rendered image and real-world data, but are currently impractical for usage with deep neural networks due to their high computation cost.

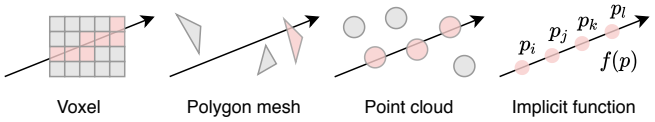

Fig. 4. Differentiable rendering algorithms differ in how the geometric information along a ray is collected and aggregated. For voxels, collecting geometric information is done by checking intersections of a ray and each voxel. For meshes, unlike non-differentiable rendering, multiple polygons have to be associated with a single ray. For point cloud, there are several ways to measure the influence of a point to a ray by pseudo-sizing. For neural implicit functions, various sampling techniques have been proposed for efficiency. Aggregation methods mainly depend on whether geometric information is treated deterministically or probabilistically, and how occlusion is handled.

Voxel-based approaches are easy to use, but require an excessive amount of memory and parameter usage. The collection and aggregation of voxels along the ray, to produce a final pixel value, is a major differentiating factor of voxel-based approaches. Despite the easiness and strength of such methods, their applicability is limited to small scenes and low resolutions. SDF-based voxel approaches allow representing surfaces more smoothly than occupancy grid-based approaches.

Point cloud-based approaches provide low computational cost while facing the sizing ambiguity. Point clouds are a natural choice for many differentiable rendering methods due to their simplicity and widespread usage with 3D sensors, but fail to capture dense surface information. Selecting a good point size and deciding the color of a rendered pixel in case of occlusion is not straightforward. Different methods have been proposed to address these issues.

Implicit representations can be viable alternatives to point clouds, voxels and meshes, but are computationally expensive when sampling points along a ray. An implicit function describes a geometry at a 3D point by its occupancy probability, transparency, or distance to surfaces. A broad range of topologies can be represented at virtually infinite resolution and with low memory cost. Despite the advantages, aggregating data on a ray may be extensive to

compute because sampling them requires evaluation of a neural network at a large number of points.

Table 3 shows the summary of the strength and limitations of different representations and rendering methods.

2.7 Open Problems

There are several problems that are not addressed by the current differentiable rendering methods.

Many applications such as 3D shape estimation and pose estimation, with the exception of 3D adversarial examples and style transfer, train neural networks by minimizing the difference between the rendered image and the target image. In such a pipeline, a rendering function and an image comparison function are usually developed separately. Therefore, a comparison function cannot leverage the rich 3D information that is fed into a rendering function. However, they can also be considered together. As mentioned in Section 2.1, the purpose of differentiable rendering is not to provide exact gradients, but to provide gradients that are useful for optimization. This could be achieved via a differentiable render-and-compare function that takes a 3D model and a target 2D image instead of a differentiable rendering function that takes a 3D model and outputs an image. One method in this direction is a differentiable projection and comparison of keypoints associated with the vertices of a 3D template shape [62]. Even though such a method is task-specific, it can be generalized by introducing the 3D information in the rendering and image comparison functions. The integration of such information is still an open research area.

Current differentiable rendering methods based on local illumination are very simple and cannot produce photorealistic images that exhibit shadows and reflections. Moreover, global illumination methods are too slow for training neural networks. On the other hand, in game engines, advances in real-time rendering methods have made it possible to render highly realistic images without using computationally-intensive path tracing. However, in the context of differentiable rendering, the area between these two is unexplored. Integrating fast, but complicated real-time rendering methods, such as shadow mapping and environment mapping, is an approach that has not been tried yet, but has the potential to improve the rendering quality.

Differentiable rendering of videos is also an interesting research direction to be explored. To achieve this, the integration of differentiable rendering with a physics simulator (to incorporate additional physical constraints) is needed. Theoretically, it should be possible to train an end-to-end pipeline that combines video data with differentiable physics simulators [63], [64], but that has not been experimented yet.

Since rendering assumes a physical model to generate an image, the rendered image will appear unrealistic if there is a large discrepancy between the physical model and the real world. On the other hand, neural rendering can produce highly realistic images as it makes almost no assumptions about the physical model. However, such a lack of assumptions may sometimes produce images that violate the physical model. For example, in neural rendering, the shape of an object can vary depending on the viewpoint. The images of objects [60] and scenes [59] rendered by neural networks do not guarantee shape consistency across different viewpoints. While ways to incorporate inductive bias about the physical world into neural rendering are important, incorporating learning-based methods into differentiable rendering is also worth considering. Humans can intuitively and instantly understand how the scene will change when the camera moves. This ability is based on past experiences, rather than calculating the value of every pixel in the brain. Therefore, such kind of learning may facilitate gradient computation. Learning-based methods have already been used for image denoising in rendering [65], [66] and efficient ray sampling [67]. These approaches may also prove useful in differentiable rendering.

3 Evaluation

Evaluation of differentiable rendering methods is not a trivial problem because rendering is a complex function. In this section, we review common practices for evaluating the algorithms and raise their problems.

A naive approach is direct gradient evaluation. For the global-illumination based methods [24], [25], [26], [27], an efficient gradient computation method when the rendering integral includes visibility terms such as object boundaries is still an open research problem. Since the goal is computing analytically-correct gradients, gradients computed by finite differences are used as ground-truth. However, the lack of a common dataset for evaluation prevents algorithms for being evaluated quantitatively. On the other hand, some papers [19], [9], [20] focus on computing *approximated gradients* for local illumination models. In that case, the gradients should be meaningful for optimization rather then being analytically correct. For this reason, finite differences cannot be used for ground-truth. They only approximate the analytically correct derivative and, therefore, may not be practical for optimization. Loper and Black [19] compare gradients computed by their method with the ones computed by finite differences and claim that the proposed approach is better. Their reasoning is that determining a good epsilon in finite differences for optimization is challenging. Visualizing gradients (without showing ground-truth) and analyzing their convergence efficiency during the optimization of the objective function is another approach used for evaluation [9], [24].

Evaluating the optimized scene parameters is another approach [19], [22], [23] that is being employed. The lack of a common dataset for comparison leads each paper to use their own for evaluating the algorithm. Instead of optimizing the parameters of a 3D scene directly, many papers train neural networks for single-view 3D object reconstruction and report the reconstruction accuracy [9], [20], [7], [8], [22], [23].

Computation time is also an important metric, especially for the ray tracing-based rendering methods. Therefore, several papers report computation time as part of the evaluation methodology [19], [24], [26].

While evaluating the local and global illumination models, two main differences can be noticed. For the global ones, as the purpose of the algorithms is calculating analytically-correct gradients, the finite differences can be used as

TABLE 3
Comparison between different algorithm types.

Representation	Type	Strengths	Limitations
Mesh	Analytical derivative	Usage of advanced forward rendering	Local minima in geometry optimization
	Approximated gradient	Usage of advanced forward rendering	Handcrafting gradient calculation
	Approximated rendering	Auto-diff support	Not precise rendering result
	Global illumination	Realistic rendering outcome	Computationally too expensive
Voxel	Occupancy / transparency	Simple, easy to optimize	Excessive memory consumption
	Signed distance function	Efficient ray trace	Not suitable for transparent volume
Point Cloud	Point cloud	Easy to render and differentiate	Pseudo-sizing, lack of surface
	RGBD image	Ordered point cloud by default	Point density reduces with the distance
Implicit	Occupancy / transparency	Simple, easy to optimize	Inside and outside is ambiguous
	Level set	Clear object boundary	Difficult to optimize

TABLE 4
Applications and representative literature of differentiable rendering.

Application	Literature
3D object reconstruction	[9], [20], [7], [8], [68], [22], [23], [69], [38], [41], [43], [62], [70], [71], [30], [72], [73], [74], [45]
Body shape estimation	[11], [10]
Hand shape estimation	[13], [75], [76]
Face reconstruction	[77]
Object/camera pose estimation	[78]
Object part segmentation	[79]
Material estimation	[17]
Light/shading estimation	[80], [81], [82], [83]
Adversarial examples	[84], [85], [86], [87], [88]
Auto-labeling	[46]
Teeth modeling	[89]

ground-truth. For the local ones, the purpose is to approximate useful gradients, therefore the optimization results should be used. However, in both cases, the lack of a common dataset prevents a fair comparison of the different algorithms. One possible solution is to create a set of toy problems which can be used to evaluate the derivative or optimization of geometry, material, light, and camera parameters. Hence, a novel evaluation methodology for differentiable rendering which is fair, accurate and meaningful is a necessity.

4 Applications

Differentiable rendering has been used to tackle various 3D-related problems of computer vision and computer graphics. In this section we review applications of differentiable rendering and discuss their limitations. Table 4 shows representative applications of DR.

4.1 Object Reconstruction

Single-view 3D object reconstruction is the task of estimating the 3D shape of an object from a single image. Unlike multi-view stereo and structure-from-motion, which use multiple images, this task is solved by machine learning rather than geometric estimation. Modern methods are able to learn high-quality 3D reconstruction from natural images, as shown in Figure 6.

A straightforward approach to single-view 3D object reconstruction is supervised learning using ground-truth 3D shapes [4], [91], [92], [93], [55], [54]. Supervised learning, which requires the annotation of 3D shapes corresponding to images, is a costly process in terms of time and labor. Such a burden can be significantly reduced by replacing the 3D annotations with 2D ones. Self-supervision is commonly employed to achieve this. As differentiable rendering allows for observing the 3D scene parameters from the image space, the input image can be used to create supervision and help train a single view 3D object reconstruction pipeline, as shown in Figure 5.

An earlier work uses the non-differentiable OpenGL renderer for single-view 3D object reconstruction based on policy gradient algorithms [94]. However, the set of the shapes to be reconstructed are limited to rough and simple ones. The advancement of voxel-based [7], [8], mesh-based [9], [22], [23], [69], point cloud-based [38] and neural implicit function-based [41], [43] differentiable rendering has significantly improved the reconstruction accuracy. Although the pipeline in Figure 5 allows for learning a high quality 3D object reconstruction, it has two major issues. First, collecting multi-view, real-world images of an object is often impossible or too costly. Second, creating accurate annotations for some scene parameters, such as camera and light, is difficult for humans.

One possible solution for the first problem is to use a single image per object instead of multi-view images. However, due to its ill-posed nature, the reconstruction of the 3D shape becomes ambiguous when a single view is used during training. To account for this, Kanazawa *et al*. [62] propose to use the reprojection error of semantic keypoints that implicitly contain rough 3D information (e.g. beak of birds) as an additional training objective. Similarly, Liu *et al*. [70] propose to use the reprojection error of semantic parts in a self-supervised manner. Although most methods do not model lighting effects, Henderson and Ferrari [71] explicitly provide light as an additional input to the renderer to use shading to help reduce the ambiguity of the reconstructed 3D shapes. Kato and Harada [20] propose to use adversarial

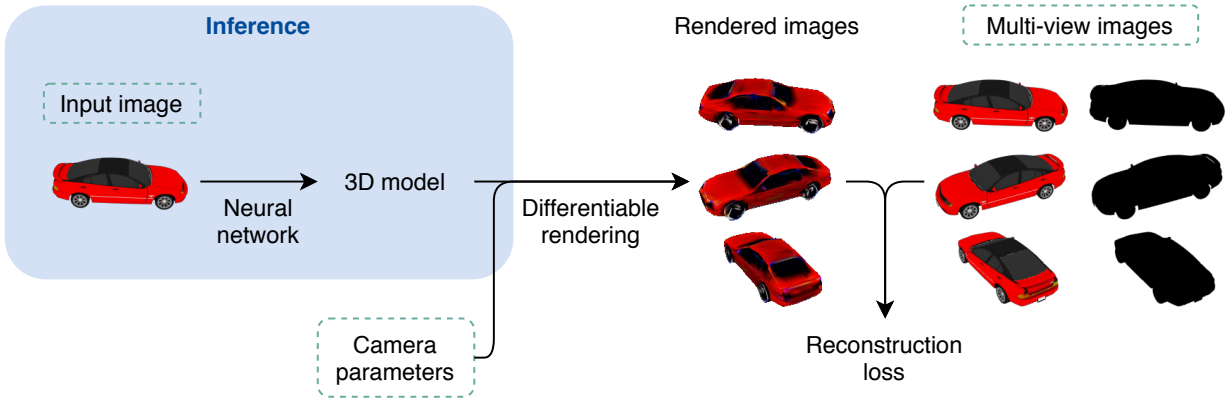

Fig. 5. Standard training pipeline of learning single-view 3D object reconstruction from 2D images. Dashed rectangles represent training data.

Fig. 6. Single-view 3D object reconstruction by Kanazawa *et al.* [90]. The leftmost column shows input images, and the rest are reconstructed objects.

Fig. 7. Human body estimation trained with silhouette and keypoint supervision by Pavlakos *et al.* [10] .

training in order to reconstruct shapes that looks realistic from any viewpoint.

For the second problem, several studies have attempted to eliminate or reduce the need for camera parameter annotations. Tulsiani *et al.* [68], Insafutdinov and Dosovitskiy [38], Henderson and Ferrari [71] propose to integrate a camera parameter estimation network into a shape estimation network and use the predicted parameters for rendering instead of using the ground-truth ones. The training objective of Henzler *et al.* [30] does not focus on improving the reconstruction quality, but rather on being able to render realistic image from random viewpoints. On the other hand, Yang *et al.* [72] propose a semi-supervised pipeline for learning the camera parameters.

Other topics in this field include learning a generative model of textured 3D objects [73], reconstruction of cars in driving scenes by leveraging 3D CAD models [74] and using differentiable rendering to fine-tune a shape estimation model trained with 3D supervision [45].

4.1.1 Limitations and Open Problems

Learning to reconstruct objects from natural image datasets of various categories is a possible research direction. Although not requiring 3D supervision is one of the advantages of learning from images, synthetic datasets are widely used in experiments. Such datasets are generated in a controlled environment in which objects are not occluded, image quality is very high and silhouette segmentation is almost noise-free. However, in real-world applications these assumptions often do not hold, which makes the synthetic dataset usage impractical for such scenarios. One of the commonly used synthetic dataset is ShapeNet [95], but other approaches [62], [70], [68], [20] use noisier natural image datasets such as Caltech-UCSD Birds-200-2011 [96] and PASCAL [68], [97], [98], [99] for training. Due to inherent limitations of the available datasets, the trainable object categories are limited to bird, cars, aeroplanes and chairs, as key point annotations are required for camera parameter estimation. Novel approaches that rely on natural images will be the key to practical applications. Alternatively, learning from videos or from the interactions of scene elements is still a challenging problem. Humans also learn 3D reconstruction based on their observations, including the temporal changes and physical interactions. This type of supervision is less studied [100], but would be an interesting topic to explore.

4.2 Human Reconstruction

4.2.1 Body Shape and Pose Reconstruction

Human body shape and pose reconstruction is a problem that has long been studied by the vision community, since it opens up the possibility for a broad range of real-world applications. While remarkable success in this area has been obtained by processing information from alternative sensors [101], [102], image sequences [103], [104] or multi-camera views [105], [106], the progress has been limited for applications that only consider monocular images, due to the 2D to 3D mapping ambiguity.

Recent advances in differentiable rendering allow for solving such ambiguities by learning prior knowledge that

can be used to recover accurate 3D shapes. Most of the methods that we have surveyed employ a pipeline in which a statistical shape model is optimized for consistency between its projection to the image plane and key 2D image observations. Loper *et al*. [107] propose *SMPL*, a skinned vertex-based model that captures correlations in human shapes across the population, to address the full human body reconstruction problems. This model is a popular statistical body shape model and it can represent a large variety of body shapes, in natural human poses, with just a small set of parameters.

Bogo *et al*. [11] propose a method that predicts the pose and 3D mesh of the human body from a single image. The core observation is that body joints hold a rich set of information that can be exploited to recover the shape. Their method predicts joint locations in the 2D space using the method of Pischuilin *et al*. [108]. The body shape is modeled using SMPL and the pose is represented by a skeleton rig with 23 joints. Each joint encodes relative rotations between body parts. To recover the 3D shape and pose, the method minimizes an objective function that accounts for five terms: three pose priors that penalize unusual bending of knees and elbows, a shape prior that ensures consistency with the SMPL model and an error term that measures the distance between the projection of the SMPL joints and estimated ones in 2D. Kanazawa *et al*. [90] further reconstruct human body shapes without using any 3D annotations.

The estimation of body shape and pose is also tackled by Pavlakos *et al*. [10] (Figure 7). They propose a pipeline in which a convolutional neural network is trained to predict both silhouettes and heatmaps corresponding to predefined keypoints. This information is used to learn pose and shape priors, which are fed into a body mesh generator (based on SMPL) that is trained to be consistent with the learned distributions. A differentiable renderer is further optimized for consistency between the mesh-to-image projection and the 2D annotations. Lassner *et al*. [109] similarly use silhouettes for reconstruction. Some recent works [110], [111] attempt to reconstruct human body shapes with clothes via a render-and-compare pipeline without using any statistical shape models and costly annotations.

Deng *et al*. [79] tackle unsupervised 3D human body parts segmentation. The approach is based on a model that parametrizes a pre-defined number of parts by their relation to the camera and deformable triangular meshes. The neural network combines the parametrized parts into the same 3D space to obtain a 3D model of the whole object in a specific pose. Then, it optimizes for the unknown parameters using the render-and-compare approach. During inference, additional poses can be retrieved by sampling the parameter space in addition to a 3D model.

4.2.2 Hand Shape and Pose Reconstruction

Efforts have gone as well into accurate reconstruction of other parts of the body, such as hand shape and pose reconstruction. One of the early attempts to learn 3D hand poses from 2D images is the work of Oberweger *et al*. [112]. Their optimization pipeline relies on the synthetic depth rendering of the hand, which is achieved through a trained neural network. Differently, several works have tried to use differentiable rendering for this purpose. Baek *et al*. [13] address these challenges by proposing a parametric model that represents 3D deformable and articulated hand meshes. They train a feature extractor that incorporates texture and shape information of the hand, as well as a pose estimator, from color images. This information is further used to regress the vertices of a hand mesh and refine its pose, through an iterative optimization process. The authors also use the trained model to perform data augmentation by generating pairs of 3D meshes, corresponding to rendered 2D masks and RGB images. A similar approach is taken by Zhang *et al*. [75], in which heatmaps corresponding to 2D keypoints act as supervision signal for estimating the parameters a 3D hand model [113]. Qualitative experiments show that their method is able to accurately recover the hand shape and pose even in the presence of severe occlusions.

Fig. 8. Unsupervised 3D face reconstruction by Wu *et al*. [114].

Zimmermann *et al*. [76] report that hand shape and pose reconstruction methods often perform well on the datasets that they are trained on, but do not always generalize to other datasets or in real-world scenarios. They introduce a multi-view hand dataset accompanied by 3D pose and shape annotations, along with a methodology that allows for accurate annotation with moderate manual intervention. Their approach uses a sparse set of 2D keypoints and semi-automatically generated segmentation masks from multiple views by fitting a deformable hand model onto them. The fitting procedure yields a 3D hand pose and shape for each view, which are used to train a 3D hand pose estimation network. At inference time, the network predicts hand poses and corresponding confidence scores on unlabeled data. This allows a human annotator to save time by annotating the least confident predictions and verifying the remaining ones.

4.2.3 Face Reconstruction

Other works have focused on face reconstruction, given the practical applications for the gaming and animation industries. Genova *et al*. [77] propose an unsupervised auto-encoder architecture which regresses the parameters of a textured 3D morphable model from image pixels. They leverage the features of a facial recognition network [115] to regress the parameters of a statistical shape model of the face, using both real-world and synthetic face images. A differentiable renderer is used to preserve feature consistency across identities, to ensure that the parameter distributions within a batch remain consistent with the morphable model and that the network can correctly interpret its output. Wu *et al*. [114] propose a completely unsupervised method for learning face reconstruction from face images, which does not rely on existing 3D morphable face models (Figure 8). Separate camera pose, depth, light and elbedo estimation networks learn to reconstruct the geomtry by leveraging shading information. A symmetry prediction network is

used to reduce the 3D geometry ambiguity, assuming that most parts of shapes are symmetric.

4.2.4 Limitations and Open Problems

Most of the presented applications revolve around the reconstruction of a 3D shape from a single image. For human body shape reconstruction, the 2D to 3D mapping ambiguity can be alleviated using a statistical model [107], [116] to learn shape priors, whose parameters are optimized via differentiable rendering. However, existing models do not diversify age by including the infants, since they are trained on adult data. Hesse *et al*. [117] report that simply scaling the body model in order to fit to infant data does not produce the desired outcome, since body proportions are different. Finding ways to generalize across different age groups and body proportions is still an open research problem.

Existing body shape datasets are also either scarce or biased towards various body types and/or ethnicities. More variety in the data is required to improve the shape models, but collecting such data is challenging due to expensive 3D scanners, privacy issues and the difficulties for humans to stand still correctly during recording [117].

Most of the body shape reconstruction methods that we have surveyed fit statistical shape models to 2D body joint locations, render the result and compare against the available observations. However, joint locations are not guaranteed to produce the optimal outcome and investigating alternative features should be considered.

Avoiding interpenetration is another challenge for human body reconstruction methods. Several works [118], [104], [119] tackle this problem by introducing an error term in the objective function which penalizes unusual poses. However, this approach does not always avoid interpenetration, since the limbs are approximated with geometric primitives for efficiency purposes. Incorporating physics-based constraints in the rendering process to help solve this problem is still an open research area.

In the animation and gaming industry, producing realistic speech animations is a challenging task. As the lip-sync performance of an actor needs to be mapped to a virtual character, determining the accurate displacement of lips is crucial. Bhat *et al*. [120] address this issue using helmet mounted cameras, which track face markers in order to find the blendshape weights of a face model. Since using markers is not always an option, other works [121], [122], [123] focus on using 2D features, depth or low-resolution images. Recently, muscle-based systems have shown potential to improve the accuracy and semantic interpretability of blendshape models. However, formulating and integrating such a model into a differentiable rendering-based pipeline is still an open research problem.

4.3 3D Adversarial Examples

Many machine learning models in computer vision are vulnerable to adversarial attacks, which are defined as inputs with visually imperceptible perturbations meant to intentionally trigger misclassification. Pixel perturbations are attractive due to their simplicity, but fail to consider the knowledge of image formation. This renders them unsuitable for real-world security applications, where an attacker does not have access to the pixel-level information. To account for these drawbacks, recent research work has been focusing on perturbing the object geometry and scene lighting parameter space.

Liu *et al*. [86] propose a method that perturbs scene lights, which are represented as spherical harmonics to filter unrealistic, high-frequency lighting while allowing usage of the analytical derivatives. On the other hand, Zeng *et al*. [84] perturb the intrinsic parameters of a scene/object. These changes in image intensities are constrained to be visually imperceptible for attacking a 2D object classifier. Xiao *et al*. [85] perturb object vertices with textures and rendered from various angles to study the vulnerable regions of the mesh. Alcorn *et al*. [87] study attacks caused by out-of-distribution error by generating unrestricted 6D poses of 3D objects and analyzing how deep neural networks responded to such unusual configurations.

Adversarial attacks can also be applied to problems unrelated to image classification. Wu *et al*. [88] analyze the vulnerabilities of Siamese networks [124] in the context of visual tracking. The authors claim that over-dependence of these trackers on similarity score, cosine window penalty and the asymmetrical region proposal network poses potential adversarial threats. They propose a differentiable rendering pipeline that generates perturbed texture maps for 3D objects, successfully lowering the tracking accuracy and even making the tracker drift off.

4.3.1 Limitations and Open Problems

Some applications are more vulnerable to adversarial attacks than others, due to limited training data being available or biased towards certain parts of the search space. For example, Alcorn *et al*. [87] report that ImageNet classifiers only label 3.09% of the 6D pose space of an object and misclassify many generated adversarial examples that are human-recognizable. As such, augmenting the training data using differentiable rendering with better photo-realistic reconstruction may help reduce the bias in the search space. Even though photo-realistic rendering has certain limitations (as explained in Section 5.3), it is worth exploring the impact of data augmentation using differentiable rendering with simple illumination models.

4.4 Other Applications

Although differentiable rendering has mainly been integrated into 3D object/human reconstruction and adversarial attacks pipelines, several works focus on different topics.

Zakharov *et al*. [46] propose an automatic annotation pipeline that recovers 9D pose (translation, rotation, scale) of cars from pre-trained off-the-shelf 2D detectors and sparse LiDAR data. They introduce a novel differentiable shape renderer based on SDF (signed distance fields) and NOCS (normalized object coordinate spaces) [127]. The renderer optimizes for geometric and physical parameters, given the learned shape priors. Beker *et al*. [125] further improve accuracy (without using LiDAR information) by leveraging monocular depth estimation and textured 3D object reconstruction. Figure 9 illustrates examples.

Several works focus on estimating the light sources from images. Nieto *et al*. [80] try to minimize the photometric error between the rendered image and the observed

Fig. 9. Automatic generation of 3D object detection labels via rendering by Zakharov *et al.* [46] (upper) and Beker *et al.* [125] (lower).

Fig. 10. Novel view syntheses by Mildenhall *et al.* [42]. Ground-truth object (first image), cropped ground-truth (second and fourth), and cropped synthesized images (third and fifth).

one, given the RGB and depth images of a scene. Unlike most previous works, which do not take cast shadows into account when estimating the illumination, their method is based on the Blinn-Phong reflection model [47].

On the other hand, Azinovic *et al*. [17] estimate the light source locations and object material properties in an indoor scene. They use 3D data and single or multiple RGB frames, along with corresponding camera poses, and employ a differentiable Monte Carlo renderer to optimize for the unknown parameters.

Differentiable rendering can also be integrated into a multi-view geometry pipeline to infer camera parameters and a dense surface model [128] or to synthesize novel views [42] (Figure 10). In addition, differentiable path tracing is currently used to capture and model human teeth [89], to estimate shader parameters [82], [83], to reconstruct objects which are not directly visible from the sensors [18], to jointly estimate the material and light parameters of a 3D scene [17], to estimate poses [78] and to analyze the lighting, shading and shadows in a scene [81].

Differentiable rendering is also useful for processing 3D models via rendered images [9], [16], [126] (Figure 11).

4.4.1 Open Problems

Several problems that have previously been studied could be improved using differentiable rendering. Examples include hand tracking [129], fabricating translucent materials [130], creating refractive caustics [131] or optimizing daylight as a part of an architectural design process [132].

Image manipulation via 3D reconstruction is a novel application of differentiable rendering. Although initial works have been limited to driving scenes [133], thanks to recent advancements in human and object reconstruction, other fields such as face image manipulation can be exploited by differentiable rendering.

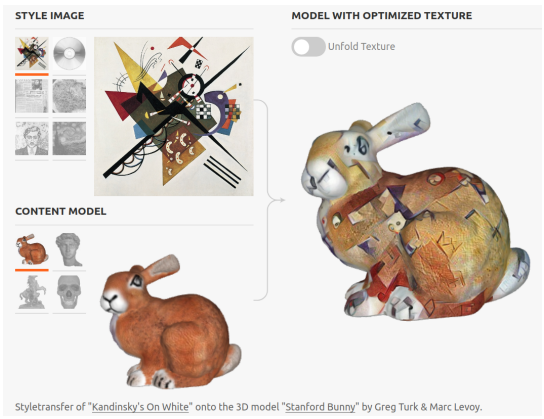

Fig. 11. Style transfer of a 3D model by Mordvintsev *et al.* [126].

5 Libraries

In this section, we present existing differentiable rendering libraries. We also briefly cover the non-differentiable ones, as they have a relatively long history and can guide the future development of the differentiable ones. We discuss the limitations and open problems as well.

We define a *differentiable rendering library* as the software that supports single or multiple differentiable rendering algorithms, implements multiple utility functions that can be used together with the rendering layer, can be integrated with existing neural network frameworks and is designed to be modular and extensible. We exclude an implementation of a single algorithm from the scope of the library definition.

5.1 Differentiable Rendering Libraries

As of second quarter of 2020, the differentiable rendering libraries available on the market are TensorFlow Graphics [134], Kaolin [135], PyTorch3D [136] and Mitsuba 2 [137]. We present a comparison of the library functionalities in Table 5. Due to continuous development, there is a possibility for the list of functionalities to be extended by the time this paper is published.

5.1.1 TensorFlow Graphics

TensorFlow Graphics is being developed by Google and is the first library in this field. It combines differentiable rendering with the geometry layers under its scope to allow for easier integration with TensorFlow-based neural network architectures. However, it only supports analytical derivative of rendering described in Section 2.1 and does not provide gradients for visibility changes. The library supports the modeling of surface and material properties, the Phong reflectance model for Lambertian surfaces with point light and spherical harmonic lighting, graph convolution support and the Chamfer distance loss. It also allows meshes to be easily visualized in TensorBoard. In addition, the Levenberg-Marquardt algorithm [138] is provided for optimization.

5.1.2 Kaolin

Kaolin is being developed by NVIDIA and is based on the PyTorch deep learning framework. Aside from differentiable rendering functionalities, the library includes implementations of various 3D neural architectures [23], [5], [139], [140], [141], [142], [143], [55] and support for triangle/quad mesh, point cloud, voxel grid and SDF representations of 3D objects. The neural architecture implementations are backed by a model zoo, which provides pretrained weights for each implementation. Kaolin preserves the original implementations of the existing differentiable rendering algorithms [9], [22], [23] and provides support for the reading and preprocessing of various datasets [144], [145], [95]. For easier integration with different datasets and architectures, the library allows the conversion between various primitive types.

5.1.3 PyTorch3D

PyTorch3D is being developed by Facebook and is based on the PyTorch deep learning framework. The core rendering algorithm and all of its dependencies are optimized through CUDA implementations for both the backward and forward passes. The rendering pipeline is composed of a rasterizer and a shader module. The rasterizer applies the 3D camera transformations and rasterization to the given shape primitives, mesh and point cloud. The output is forwarded to the shader component which applies lighting, interpolates textures and calculates reflectance shadings including hard/soft Phong [47], hard/soft Gouraud [146], hard flat and soft silhouette shading to create the final image. Besides the core rendering pipeline, PyTorch3D also supports several 3D loss functions including the Chamfer distance, mesh edge loss [142], Laplacian regularization loss [142] and normal consistency loss [142], along with graph convolutions. As a unique functionality, PyTorch3D can support heterogeneous batching of 3D data which allows usage of different sized meshes as input of the renderer. Such functionality is essential to represent 3D object efficiently with different level of details to find the balance between the vertex count and the visual quality depending on the application.

5.1.4 Mitsuba 2

Mitsuba 2 proposes a novel approach for efficient high-performance rendering with auto-differentiation. It is designed as an efficient, internally portable for different compute units for multi-purpose usage. 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. Mitsuba 2 demonstrates its effectiveness and ease of use for tackling challenging problems: the Markov Chain Monte Carlo (MCMC) rendering technique is used to explore nearby light paths coherently to ensure faster convergence, a method for creating gradient-index optics which focuses incident illumination into caustics, and a method for reconstructing the parameters of a heterogeneous participating medium from multiple images.

Mitsuba 2, different from other libraries, focuses mainly on speed and efficiency through three guiding principles: no duplication, unobtrusiveness and modularity. It performs symbolic arithmetic, which delays the evaluation of a kernel until the variable is accessed. For the auto-differentiation-based algorithms, all the calculated variables and intermediate values are kept in memory. Especially for the inter-reflection scenes, such an approach consumes a significant amount of memory, which limits the rendered image size and the 3D object polygon count that can be processed at a time. To overcome this problem, Mitsuba 2 simplifies the computation graph periodically before each JIT compilation pass, using vertex elimination [147], [148].

Besides the efficient implementation and design, Mitsuba 2 proposes two novel MCMC schemes, *Coherent Pseudo-Marginal MLT* and *Multiple-Try Metropolis*. Coherent Pseudo-Marginal MLT algorithm is based on idea sampling a modified target function π that is the result of convolving the path contribution function f with a Gaussian kernel G. Such an approach produces coherent bundles of rays at each iteration and enhances the memory access speed assuming that the coherent ray's physical memory locations are close to each other. Multiple-Try Metropolis(MTM) is a modified version of the previous work, which integrates MTM into a vectorized MTL [149], [150]. The novel MTM algorithm generates a set of N proposals to choose from at each iteration instead of the conventional random walk.

5.1.5 Comparison

All the libraries are optimized either through deep learning frameworks, which are well established for CPU/GPU agnostic coding with Python, or through custom C++/CUDA access. Mitsuba 2 is the only library that is based on a custom auto-differentiation module, Enoki. It also provides Python bindings on top of the C++ core, while other libraries benefit from the auto-differentiation capabilities of the PyTorch/TensorFlow libraries in Python.

Tensorflow Graphics and Kaolin aim to integrate novel algorithms as 3rd party into their structure by preserving the original implementation. Such collection-oriented design choice reduces the required work of integrating the novel algorithms and allows easy comparison between different algorithms. However, as the rendering pipeline of each algorithm and their optimization levels are not centralized, the learning curve of a new algorithm, integration between multiple algorithms and the rendering speed are highly susceptible to vary significantly. On the other hand, PyTorch3D and Mitsuba 2 aim to provide a customizable pipeline where users can extend the capabilities of each submodule of it. Such customization-oriented design choice allow novel algorithms to be integrated and developed using extensible components for lighting, shading, texturing and blending. While such design choice allows users to flexibly manipulate the renderer components and to experiment new research ideas efficiently, it requires an extra effort to integrate 3rd party algorithms if they are released as standalone implementations.

Besides the rendering capabilities of each library, Kaolin's ready to use state-of-art architecture implementations, along with pretrained model zoo, reduces the re-implementation and re-training cost. Also, having dataset loading and preprocessing functions is a strong advantage compared to other alternatives. On the other hand, the

native support for the OBJ and PLY formats in PyTorch3D and Mitsuba 2 reduces the dependency on other libraries for I/O operations. The integration between TensorBoard and TensorFlow Graphics allows users to easily visualize rendered output. On the other hand, Kaolin provides several functions for visualizing meshes, voxels, and point clouds through the *pptk* backend.

Besides Mitsuba 2, each library focuses on rasterization instead of ray tracing. This allows them to be easily usable with synthetic datasets and simple scenes without complex reflectance. Considering that the number of supported shading algorithms and surface materials, real world scene applications of existing libraries have so far been limited.

5.2 Non-Differentiable Rendering

Several non-differentiable rasterizer-based and ray tracing-based rendering libraries have been developed over the last few decades.

Rasterization-based libraries are preferred for applications where the rendering speed is more important than the visual quality. They commonly rely on the Direct3D [151], OpenGL [152] or Vulkan [153] APIs as backend instead of providing custom implementations. These APIs support common functionalities such as easy GPU access, ready-to-use rasterization pipeline and z-buffer algorithms. They implement their own shader language to allow developers to extend their capabilities. However, due to the large development cost incurred by the direct use of those APIs, it is also common to use the high-level wrappers Unity [154] or Unreal Engine [155]. The game engines provide additional utility functions and allow designing interfaces. This allows easier portability of the rendering software across multiple hardware devices whose direct access API may be significantly different. It also reduces the knowledge requirement of a developer to hardware-specific content. Major strengths of high-level libraries are the easy integration of complex lights and shadows, animation functions, advanced I/O system for saving/loading various assets and scene information for reducing the research and development cost. The engines also support multiple programming languages for development, which helps reduce the learning curve required when writing production-level code. Lastly, they provide GUIs for easy interaction and visualization. This speeds up the development process and helps developers notice possible bugs early on, by visualizing intermediate results.

Photo-realistic rendering by ray tracing consumes more resources than rasterization. Hence, ray tracing-based libraries are preferred for applications where the visual quality is more important than the rendering speed. Arnold [156], [157], V-Ray [158] and RenderMan [159], [160] are popular ray tracing-based libraries. They are integrated into digital content creation (DCC) tools such as Maya [161] and 3ds Max [162] to provide end-user GUI access. For the materials and shaders, RenderMan provides its own shader language, RSL [160], as an extension to the editing capabilities through GUI. These libraries support voxel and parametric surfaces as shape primitives, in addition to meshes. Similar to rasterization, industrial ray tracing libraries are based on several low-level APIs such as Embree [163], OSPRay [164], OptiX [165] and Iray [166]. Embree and OSPRay are optimized for Intel CPUs by using SIMD instructions. While Embree provides the ray tracing acceleration kernels, OSPRay contains extra functionalities such as volume rendering, global illumination, easier industry adoption, distributed computation and support for multiple shape primitives. On the other hand, the GPU-based libraries OptiX and Iray also follow the same structure. While OptiX provides the core kernel functionalities, Iray adds extra features like AI denoising, RTX hardware support, multi-GPU setup and easier third-party integration.

5.3 Limitations and Open Problems

Despite the emergent development of differentiable rendering libraries, there are still limitations and possible advancement directions to be addressed. We list some of them below:

Support for embedded environments is limited. Optimizing an algorithm is essential to enable it to run on embedded environments where hardware resources and processing power are limited. For NVIDIA GPUs, TensorRT[1] has been developed for optimizing a trained network's inference speed. However, a similar functionality for optimizing differentiable rendering algorithms for embedded environments is still limited.

Current libraries do not provide a standard interface for extensions. Current differentiable rendering libraries either require new algorithms to be implemented from scratch or provide basic extension layers through a plugin-based architecture. Providing extension capabilities which are common across the frameworks, similar to the non-differentiable rendering community, would be an added value.

Existing implementations have limited functionalities. Differentiable rendering libraries are designed mainly for computer vision researchers and engineers. Functionalities that currently exist in the non-differentiable ones, such as integration with DCC tools, support for primitives other than triangles, various output formats such as high-dynamic range images, meta-data/animation rendering, are currently lacking. We believe that adding more functionalities required for the game and movie industries would enable the development of new types of applications. Also, existing libraries (except for Kaolin) do not provide ready-to-use support for the datasets which are commonly used in differentiable rendering research. This, in turn, leads to repetitive implementation of dataset management, while increasing the development costs.

Support for light and material models is limited. As light and surface material quality are essential for photo-realistic RGB rendering, the development of advanced light support for both rasterization and ray tracing-based libraries is limited and not unified. Such limitations of the light models hinder the realistic appearance of the rendered RGB image and, therefore, the quality of the supervision signal.

Limited debugging and benchmarking support. Debugging tools and benchmarking support are crucial for efficient development and research. Existing libraries benefit from language-specific and/or third-party debugging tools

[1] https://developer.nvidia.com/tensorrt

TABLE 5
Comparison between existing differentiable rendering libraries

	PyTorch 3D	Kaolin	TensorFlow Graphics	Mitsuba 2
Core implementation	PyTorch/CUDA	PyTorch/CUDA	TensorFlow/C++/GPU	C++/CUDA Python bindings
Supported primitives	Triangle mesh Point cloud	Triangle/quad mesh Point cloud Voxel grid SDF	Triangle mesh Point cloud	Triangle mesh Custom
Differentiable rendering algorithms	Soft Rasterizer (extendable)	NMR Soft Rasterizer DIB-Renderer	Analytical derivatives	Loubet *et al.*
Rendering Method	Rasterization	Rasterization	Rasterization	Ray tracing
3D operations	Graph convolutions 3D transformations Point Cloud Operations (Umeyama, ICP, KNN)	Primitive conversions 3D transformations	Graph convolutions 3D transformations	3D transformations
Shader support	Hard/soft Phong Hard/soft Gouraud Hard flat Soft silhouette	Phong	Phong	Wide range BSDF
Lighting support	Point Directional	Ambient Directional	Point Spherical harmonic	Area Point Spot Constant environment Environment map Directional
Loss functions	Chamfer Distance Mesh edge Laplacian smoothing Normal consistency	Chamfer distance Directed distance Mesh Laplacian	Chamfer distance	Not supported
Camera support	Perspective Ortographic	Perspective Ortographic	Perspective Ortographic Quadratic distortion	Perspective (pinole, thin lens) Irradiance meter Radiance meter
Data I/O support	OBJ PLY	External	External	OBJ PLY
Data support	ShapeNet R2N2	ScanNet ModelNet ShapeNet	Not supported	Not supported
Architectures	Not supported	DIB-Renderer PointNet PointNet++ GResNet 3D-GAN Pixel2Mesh GEOMetrics Occupancy Networks	Not supported	Not supported
Other functionalities	Heterogeneous batches	Model zoo Visualization	TensorBoard (mesh)	Scene file support Extensible by plugin
Version	0.2	0.1.0	1.0.0	2.0.0

for benchmarking, if any. A benchmarking and debugging tool specifically designed for differentiable rendering, which aims to detect both rendering and gradient calculation inaccuracies, is yet to be developed.

Model sharing across different libraries is currently not possible. Support for model sharing across multiple deep learning libraries is another limitation. Recently, the ONNX format[2] has been developed to allow for easy sharing of trained models. However, due to the implementation differences across the different libraries, it is still not possible to export a neural network model which contains differentiable rendering layers and use it in a different library for inference through the ONNX interface.

6 Conclusion

This paper presented an overview of the current state of differentiable rendering research - popular algorithms and data representation techniques, as well as evaluation metrics and common practices. It also introduced applications that make use of DR-based techniques, covering a broad range of fields such as 3D object/body shape and pose reconstruction, adversarial attacks, auto labeling or light source estimation. Several libraries that are commonly used to develop new differentiable rendering-based methods were presented and compared. Finally, open problems for algorithms, applications and libraries were discussed.

Although differentiable rendering is a novel field, it is rapidly maturing, aided by the continuous development of new tools meant to simplify its usage. This will enable more researchers to develop novel neural network models that understand 3D information from 2D image data. Consequently, the performance of various applications can be improved, while the need for 3D data collection and annotation can be reduced. Once mature enough to be deployed to embedded devices, we will likely start seeing differentiable rendering-based methods solving problems with real-time constraints (such as autonomous driving). The times that we live are exciting and the advancements in deep neural networks help us tackle and solve everyday problems. The best is yet to come.

Acknowledgments

This work was supported by Toyota Research Institute Advanced Development, Inc. We would like to thank the PyTorch3D developers for their insightful comments and suggestions.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *ICLR*, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.

[4] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction," in *ECCV*, 2016.

[5] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *CVPR*, 2017.

[6] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," in *CVPR*, 2018.

[7] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee, "Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision," in *NeurIPS*, 2016.

[8] S. Tulsiani, T. Zhou, A. A. Efros, and J. Malik, "Multi-view supervision for single-view reconstruction via differentiable ray consistency," in *CVPR*, 2017.

[9] H. Kato, Y. Ushiku, and T. Harada, "Neural 3D Mesh Renderer," in *CVPR*, 2018.

[10] G. Pavlakos, L. Zhu, X. Zhou, and K. Daniilidis, "Learning to estimate 3d human pose and shape from a single color image," in *CVPR*, 2018.

[11] F. Bogo, A. Kanazawa, C. Lassner, P. Gehler, J. Romero, and M. J. Black, "Keep it SMPL: Automatic Estimation of 3D Human Pose and Shape from a Single Image," in *ECCV*, 2016.

[12] L. Ge, Z. Ren, Y. Li, Z. Xue, Y. Wang, J. Cai, and J. Yuan, "3D Hand Shape and Pose Estimation From a Single RGB Image," in *CVPR*, 2019.

[13] S. Baek, K. I. Kim, and T.-K. Kim, "Pushing the Envelope for RGB-Based Dense 3D Hand Pose Estimation via Neural Rendering," in *CVPR*, 2019.

[14] K. Genova, F. Cole, A. Maschinot, A. Sarna, D. Vlasic, and W. T. Freeman, "Unsupervised Training for 3D Morphable Model Regression," in *CVPR*, 2018.

[15] M. Bao, M. Cong, S. Grabli, and R. Fedkiw, "High-Quality Face Capture Using Anatomical Muscles," in *CVPR*, 2019.

[16] H.-T. D. Liu, M. Tao, and A. Jacobson, "Paparazzi: Surface Editing by way of Multi-View Image Processing," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 221:1–221:11, 2018.

[17] D. Azinovic, T.-M. Li, A. Kaplanyan, and M. Niessner, "Inverse Path Tracing for Joint Material and Lighting Estimation," in *CVPR*, 2019.

[18] C.-Y. Tsai, A. C. Sankaranarayanan, and I. Gkioulekas, "Beyond Volumetric Albedo–A Surface Optimization Framework for Non-Line-Of-Sight Imaging," in *CVPR*, 2019.

[19] M. M. Loper and M. J. Black, "OpenDR: An approximate differentiable renderer," in *ECCV*, 2014.

[20] H. Kato and T. Harada, "Learning view priors for single-view 3d reconstruction," in *CVPR*, 2019.

[21] H. Rhodin, N. Robertini, C. Richardt, H.-P. Seidel, and C. Theobalt, "A Versatile Scene Model with Differentiable Visibility Applied to Generative Pose Estimation," in *ICCV*, 2015.

[22] S. Liu, W. Chen, T. Li, and H. Li, "Soft Rasterizer: A Differentiable Renderer for Image-Based 3D Reasoning," in *ICCV*, 2019.

[23] W. Chen, J. Gao, H. Ling, E. Smith, J. Lehtinen, A. Jacobson, and S. Fidler, "Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer," in *NeurIPS*, 2019.

[24] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen, "Differentiable monte carlo ray tracing through edge sampling," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 222:1–222:11, 2018.

[25] C. Zhang, L. Wu, C. Zheng, I. Gkioulekas, R. Ramamoorthi, and S. Zhao, "A differential theory of radiative transfer," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 227:1–227:16, 2019.

[26] G. Loubet, N. Holzschuch, and W. Jakob, "Reparameterizing discontinuous integrands for differentiable rendering," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 228:1–228:14, 2019.

[27] C. Zhang, B. Miller, K. Yan, I. Gkioulekas, and S. Zhao, "Path-Space Differentiable Rendering," *ACM Trans. Graph.*, vol. 39, no. 4, pp. 143:1–143:19, 2020.

[28] M. Nimier-David, S. Speierer, B. Ruiz, and W. Jakob, "Radiative backpropagation: An adjoint method for lightning-fast differentiable rendering," *ACM Trans. Graph.*, vol. 39, no. 4, 2020.

[29] S. Tulsiani, T. Zhou, A. A. Efros, and J. Malik, "Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency," *TPAMI*, 2019.

[30] P. Henzler, N. J. Mitra, and T. Ritschel, "Escaping Plato's Cave: 3D Shape From Adversarial Rendering," in *ICCV*, 2019.

[31] S. Lombardi, T. Simon, J. Saragih, G. Schwartz, A. Lehrmann, and Y. Sheikh, "Neural volumes: Learning dynamic renderable volumes from images," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 65:1–65:14, 2019.

[2] https://onnx.ai

[32] Y. Jiang, D. Ji, Z. Han, and M. Zwicker, "SDFDiff: Differentiable Rendering of Signed Distance Fields for 3D Shape Optimization," in *CVPR*, 2020.

[33] R. Roveri, A. C. ztireli, I. Pandele, and M. H. Gross, "PointProNets: Consolidation of Point Clouds with Convolutional Neural Networks," *CGF*, vol. 37, no. 2, pp. 87–99, 2018.

[34] W. Yifan, F. Serena, S. Wu, C. ztireli, and O. Sorkine-Hornung, "Differentiable Surface Splatting for point-based Geometry Processing," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 230:1–230:14, 2019.

[35] O. Wiles, G. Gkioxari, R. Szeliski, and J. Johnson, "SynSin: End-to-end View Synthesis from a Single Image," in *CVPR*, 2020.

[36] C.-H. Lin, C. Kong, and S. Lucey, "Learning Efficient Point Cloud Generation for Dense 3D Object Reconstruction," in *AAAI*, 2018.

[37] L. Li, S. Zhu, H. Fu, P. Tan, and C.-L. Tai, "End-to-End Learning Local Multi-view Descriptors for 3D Point Clouds," in *CVPR*, 2020.

[38] E. Insafutdinov and A. Dosovitskiy, "Unsupervised Learning of Shape and Pose with Differentiable Point Clouds," in *NeurIPS*, 2018.

[39] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *CVPR*, 2017.

[40] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *CVPR*, 2017.

[41] S. Liu, S. Saito, W. Chen, and H. Li, "Learning to infer implicit surfaces without 3d supervision," in *NeurIPS*, 2019.

[42] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," in *ECCV*, 2020.

[43] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, "Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision," in *CVPR*, 2020.

[44] L. Yariv, M. Atzmon, and Y. Lipman, "Universal Differentiable Renderer for Implicit Neural Representations," *arXiv:2003.09852*, 2020.

[45] S. Liu, Y. Zhang, S. Peng, B. Shi, M. Pollefeys, and Z. Cui, "DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing," in *CVPR*, 2019.

[46] S. Zakharov, W. Kehl, A. Bhargava, and A. Gaidon, "Autolabeling 3D Objects with Differentiable Rendering of SDF Shape Priors," in *CVPR*, 2020.

[47] J. F. Blinn, "Models of Light Reflection for Computer Synthesized Pictures," in *SIGGRAPH*, 1977.

[48] J. Lambert, "Photometria," 1760.

[49] R. Ramamoorthi and P. Hanrahan, "An Efficient Representation for Irradiance Environment Maps," *ACM Trans. Graph.*, vol. 20, no. 3, pp. 497–500, 2001.

[50] J. T. Kajiya, "The Rendering Equation," *ACM Trans. Graph.*, vol. 20, no. 4, pp. 143–150, 1986.

[51] E. Veach, *Robust Monte Carlo Methods for Light Transport Simulation*. Stanford University PhD thesis, 1997.

[52] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu, "Spatial Transformer Networks," in *NeurIPS*, 2015.

[53] X. Roynard, J.-E. Deschaud, and F. Goulette, "Classification of point cloud scenes with multiscale voxel deep network," *arXiv:1804.03583*, 2018.

[54] Z. Chen and H. Zhang, "Learning implicit fields for generative shape modeling," in *CVPR*, 2019.

[55] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," in *CVPR*, 2019.

[56] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation," in *CVPR*, 2019.

[57] M. Oechsle, L. Mescheder, M. Niemeyer, T. Strauss, and A. Geiger, "Texture Fields: Learning Texture Representations in Function Space," in *ICCV*, 2019.

[58] S. A. Eslami, D. J. Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor *et al.*, "Neural scene representation and rendering," *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018.

[59] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, "Scene representation networks: Continuous 3D-structure-aware neural scene representations," in *NeurIPS*, 2019.

[60] T. Nguyen-Phuoc, C. Li, L. Theis, C. Richardt, and Y.-L. Yang, "Hologan: Unsupervised learning of 3d representations from natural images," in *ICCV*, 2019.

[61] A. Tewari, O. Fried, J. Thies, V. Sitzmann, S. Lombardi, K. Sunkavalli, R. Martin-Brualla, T. Simon, J. Saragih, M. Niener, R. Pandey, S. Fanello, G. Wetzstein, J.-Y. Zhu, C. Theobalt, M. Agrawala, E. Shechtman, D. B. Goldman, and M. Zollhfer, "State of the Art on Neural Rendering," *CGF*, 2020.

[62] A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik, "Learning category-specific mesh reconstruction from image collections," in *ECCV*, 2018.

[63] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-End Differentiable Physics for Learning and Control," in *NeurIPS*, 2018.

[64] Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik, "ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics," in *ICRA*, 2019.

[65] M. Gharbi, T.-M. Li, M. Aittala, J. Lehtinen, and F. Durand, "Sample-based monte carlo denoising using a kernel-splatting network," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 125:1–125:12, 2019.

[66] M. Kettunen, E. Härkönen, and J. Lehtinen, "Deep Convolutional Reconstruction for Gradient-Domain Rendering," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 126:1–126:12, 2019.

[67] T. Müller, B. Mcwilliams, F. Rousselle, M. Gross, and J. Novák, "Neural Importance Sampling," *ACM Trans. Graph.*, vol. 38, no. 5, pp. 145:1–145:19, 2019.

[68] S. Tulsiani, A. A. Efros, and J. Malik, "Multi-view consistency as supervisory signal for learning shape and pose prediction," in *CVPR*, 2018.

[69] F. Petersen, A. H. Bermano, O. Deussen, and D. Cohen-Or, "Pix2Vex: Image-to-Geometry Reconstruction using a Smooth Differentiable Renderer," *arXiv:1903.11149*, 2019.

[70] X. Li, S. Liu, K. Kim, S. De Mello, V. Jampani, M.-H. Yang, and J. Kautz, "Self-supervised Single-view 3D Reconstruction via Semantic Consistency," in *ECCV*, 2020.

[71] P. Henderson and V. Ferrari, "Learning Single-Image 3D Reconstruction by Generative Modelling of Shape, Pose and Shading," *IJCV*, vol. 128, no. 4, pp. 835–854, 2020.

[72] G. Yang, Y. Cui, S. Belongie, and B. Hariharan, "Learning Single-View 3D Reconstruction with Limited Pose Supervision," in *ECCV*, 2018.

[73] P. Henderson, V. Tsiminaki, and C. H. Lampert, "Leveraging 2D Data to Learn Textured 3D Mesh Generation," in *CVPR*, 2020.

[74] A. Kundu, Y. Li, and J. M. Rehg, "3d-rcnn: Instance-level 3d object reconstruction via render-and-compare," in *CVPR*, 2018.

[75] X. Zhang, Q. Li, H. Mo, W. Zhang, and W. Zheng, "End-to-End Hand Mesh Recovery From a Monocular RGB Image," in *ICCV*, 2019.

[76] C. Zimmermann, D. Ceylan, J. Yang, B. Russell, M. Argus, and T. Brox, "FreiHAND: A Dataset for Markerless Capture of Hand Pose and Shape from Single RGB Images," in *ICCV*, 2019.

[77] K. Genova, F. Cole, A. Maschinot, A. Sarna, D. Vlasic, and W. T. Freeman, "Unsupervised training for 3d morphable model regression," in *CVPR*, 2018.

[78] H. Rhodin, N. Robertini, C. Richardt, H.-P. Seidel, and C. Theobalt, "A Versatile Scene Model with Differentiable Visibility Applied to Generative Pose Estimation," in *ICCV*, 2015.

[79] B. Deng, S. Kornblith, and G. Hinton, "Cerberus: A Multi-headed Derenderer," *arXiv:1905.11940*, 2019.

[80] G. Nieto, S. Jiddi, and P. Robert, "Robust Point Light Source Estimation Using Differentiable Rendering," *arXiv:1812.04857*, 2018.

[81] R. Ramamoorthi, D. Mahajan, and P. Belhumeur, "A First-Order Analysis of Lighting, Shading, and Shadows," *ACM Trans. Graph.*, vol. 26, no. 1, pp. 2:1–2:21, 2007.

[82] P. Khungurn, D. Schroeder, S. Zhao, K. Bala, and S. Marschner, "Matching Real Fabrics with Micro-Appearance Models," *ACM Trans. Graph.*, vol. 35, no. 1, pp. 1:1–1:26, 2016.

[83] S. Zhao, L. Wu, F. Durand, and R. Ramamoorthi, "Downsampling scattering parameters for rendering anisotropic media," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 166:1–166:11, 2016.

[84] X. Zeng, C. Liu, Y.-S. Wang, W. Qiu, L. Xie, Y.-W. Tai, C.-K. Tang, and A. L. Yuille, "Adversarial attacks beyond the image space," in *CVPR*, 2019.

[85] C. Xiao, D. Yang, B. Li, J. Deng, and M. Liu, "MeshAdv: Adversarial Meshes for Visual Recognition," in *CVPR*, 2019.

[86] H.-T. D. Liu, M. Tao, C.-L. Li, D. Nowrouzezahrai, and A. Jacobson, "Beyond Pixel Norm-Balls: Parametric Adversaries using an Analytically Differentiable Renderer," in *ICLR*, 2019.

[87] M. A. Alcorn, Q. Li, Z. Gong, C. Wang, L. Mai, W.-S. Ku, and A. Nguyen, "Strike (With) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects," in *CVPR*, 2019.

[88] X. Wu, X. Wang, X. Zhou, and S. Jian, "STA: Adversarial Attacks on Siamese Trackers," *arXiv:1909.03413*, 2019.

[89] Z. Velinov, M. Papas, D. Bradley, P. Gotardo, P. Mirdehghan, S. Marschner, J. Novk, and T. Beeler, "Appearance Capture and Modeling of Human Teeth," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 207:1–207:13, 2018.

[90] A. Kanazawa, M. J. Black, D. W. Jacobs, and J. Malik, "End-to-end recovery of human shape and pose," in *CVPR*, 2018.

[91] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3d object reconstruction from a single image," in *CVPR*, 2017.

[92] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, "A papier-mâché approach to learning 3d surface generation," in *CVPR*, 2018.

[93] N. Wang, Y. Zhang, Z. Li, Y. Fu, H. Yu, W. Liu, X. Xue, and Y.-G. Jiang, "Pixel2Mesh: 3D Mesh Model Generation via Image Guided Deformation," *TPAMI*, 2020.

[94] D. J. Rezende, S. A. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, and N. Heess, "Unsupervised learning of 3d structure from images," in *NeurIPS*, 2016.

[95] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," *arXiv:1512.03012*, 2015.

[96] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," California Institute of Technology, Tech. Rep. CNS-TR-2011-001, 2011.

[97] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.

[98] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *IJCV*, vol. 88, no. 2, pp. 303–338, 2010.

[99] Y. Xiang, R. Mottaghi, and S. Savarese, "Beyond PASCAL: A benchmark for 3D object detection in the wild," in *WACV*, 2014.

[100] D. Novotny, D. Larlus, and A. Vedaldi, "Learning 3D Object Categories by Looking Around Them," in *ICCV*, 2017.

[101] T. von Marcard, B. Rosenhahn, M. J. Black, and G. Pons-Moll, "Sparse Inertial Poser: Automatic 3D Human Pose Estimation from Sparse IMUs," *CGF*, vol. 36, no. 2, pp. 349–360, 2017.

[102] A. Weiss, D. Hirshberg, and M. J. Black, "Home 3d body scans from noisy image and range data," in *ICCV*, 2011.

[103] W. Xu, A. Chatterjee, M. Zollhöfer, H. Rhodin, D. Mehta, H.-P. Seidel, and C. Theobalt, "MonoPerfCap: Human Performance Capture From Monocular Video," *ACM Trans. Graph.*, vol. 37, no. 2, pp. 27:1–27:15, 2018.

[104] T. Alldieck, M. Kassubeck, B. Wandt, B. Rosenhahn, and M. Magnor, "Optical Flow-based 3D Human Motion Estimation from Monocular Video," in *German Conference on Pattern Recognition*, 2017.

[105] H. Rhodin, N. Robertini, D. Casas, C. Richardt, H.-P. Seidel, and C. Theobalt, "General Automatic Human Shape and Motion Capture Using Volumetric Contour Cues," in *ECCV*, 2016.

[106] Y. Huang, F. Bogo, C. Lassner, A. Kanazawa, P. V. Gehler, J. Romero, I. Akhter, and M. J. Black, "Towards Accurate Marker-Less Human Shape and Pose Estimation over Time," in *3DV*, 2017.

[107] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A Skinned Multi-Person Linear Model," *ACM Trans. Graph.*, vol. 34, no. 6, pp. 248:1–248:16, 2015.

[108] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. V. Gehler, and B. Schiele, "DeepCut: Joint Subset Partition and Labeling for Multi Person Pose Estimation," in *CVPR*, 2016.

[109] C. Lassner, J. Romero, M. Kiefel, F. Bogo, M. J. Black, and P. V. Gehler, "Unite the People: Closing the Loop Between 3D and 2D Human Representations," in *CVPR*, 2017.

[110] R. Natsume, S. Saito, Z. Huang, W. Chen, C. Ma, H. Li, and S. Morishima, "Siclope: Silhouette-based clothed people," in *CVPR*, 2019.

[111] S. Saito, Z. Huang, R. Natsume, S. Morishima, A. Kanazawa, and H. Li, "Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization," in *ICCV*, 2019.

[112] M. Oberweger, P. Wohlhart, and V. Lepetit, "Training a feedback loop for hand pose estimation," in *ICCV*, 2015.

[113] J. Romero, D. Tzionas, and M. J. Black, "Embodied Hands: Modeling and Capturing Hands and Bodies Together," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 245:1–245:17, 2017.

[114] S. Wu, C. Rupprecht, and A. Vedaldi, "Unsupervised learning of probably symmetric deformable 3d objects from images in the wild," in *CVPR*, 2020.

[115] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," in *CVPR*, 2015.

[116] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, "SCAPE: shape completion and animation of people," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 408–416, 2005.

[117] N. Hesse, S. Pujades, M. Black, M. Arens, U. Hofmann, and S. Schroeder, "Learning and Tracking the 3D Body Shape of Freely Moving Infants from RGB-D sequences," *TPAMI*, 2019.

[118] E. Gundogdu, V. Constantin, A. Seifoddini, M. Dang, M. Salzmann, and P. Fua, "Garnet: A two-stream network for fast and accurate 3d cloth draping," in *ICCV*, 2019.

[119] G. Pavlakos, V. Choutas, N. Ghorbani, T. Bolkart, A. A. Osman, D. Tzionas, and M. J. Black, "Expressive Body Capture: 3D Hands, Face, and Body from a Single Image," in *CVPR*, 2019.

[120] K. S. Bhat, R. Goldenthal, Y. Ye, R. Mallet, and M. Koperwas, "High Fidelity Facial Animation Capture and Retargeting with Contours," in *Symposium on computer animation*, 2013.

[121] C. Cao, Y. Weng, S. Lin, and K. Zhou, "3d shape regression for real-time facial animation," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 41:1–41:10, 2013.

[122] Y.-L. Chen, H.-T. Wu, F. Shi, X. Tong, and J. Chai, "Accurate and Robust 3D Facial Capture Using a Single RGBD Camera," in *ICCV*, 2013.

[123] J. Thies, M. Zollhofer, M. Stamminger, C. Theobalt, and M. Nießner, "Face2Face: Real-Time Face Capture and Reenactment of RGB Videos," in *CVPR*, 2016.

[124] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *ECCV*, 2016.

[125] D. Beker, H. Kato, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon, "Self-Supervised Differentiable Rendering for Monocular 3D Object Detection," in *ECCV*, 2020.

[126] A. Mordvintsev, N. Pezzotti, L. Schubert, and C. Olah, "Differentiable image parameterizations," *Distill*, 2018.

[127] H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, and L. J. Guibas, "Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation," in *CVPR*, 2019.

[128] V. N. Smelyansky, R. D. Morris, F. O. Kuehnel, D. A. Maluf, and P. Cheeseman, "Dramatic Improvements to Feature Based Stereo," in *ECCV*, 2002.

[129] M. de La Gorce, N. Paragios, and D. J. Fleet, "Model-Based Hand Tracking with Texture, Shading and Self-Occlusions," in *CVPR*, 2008.

[130] M. Papas, C. Regg, W. Jarosz, B. Bickel, P. Jackson, W. Matusik, S. Marschner, and M. Gross, "Fabricating translucent materials using continuous pigment mixtures," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 146:1–146:12, 2013.

[131] Y. Schwartzburg, R. Testuz, A. Tagliasacchi, and M. Pauly, "High-Contrast Computational Caustic Design," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 74:1–74:11, 2014.

[132] M. Andersen, S. Kleindienst, L. Yi, J. Lee, M. Bodart, and B. Cutler, "An Intuitive Daylighting Performance Analysis and Optimization Approach," *Building Research & Information*, vol. 36, no. 6, pp. 593–607, 2008.

[133] S. Yao, T. M. Hsu, J.-Y. Zhu, J. Wu, A. Torralba, B. Freeman, and J. Tenenbaum, "3d-aware scene manipulation via inverse graphics," in *NeurIPS*, 2018.

[134] J. Valentin, C. Keskin, P. Pidlypenskyi, A. Makadia, A. Sud, and S. Bouaziz, "TensorFlow Graphics: Computer Graphics Meets Deep Learning," 2019.

[135] K. J., E. Smith, J.-F. Lafleche, C. Fuji Tsang, A. Rozantsev, W. Chen, T. Xiang, R. Lebaredian, and S. Fidler, "Kaolin: A PyTorch Library for Accelerating 3D Deep Learning Research," *arXiv:1911.05063*, 2019.

[136] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari, "PyTorch3D," https://github.com/facebookresearch/pytorch3d, 2020.

[137] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob, "Mitsuba 2: A Retargetable Forward and Inverse Renderer," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 203:1–203:17, 2019.
[138] J. J. Moré, "The levenberg-marquardt algorithm: Implementation and theory," in *Numerical Analysis*, ser. Lecture Notes in Mathematics, G. Watson, Ed. Springer Berlin Heidelberg, 1978, vol. 630, pp. 105–116.
[139] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *NeurIPS*, 2017.
[140] J. Zhang, "GResNet: Graph Residuals for Reviving Deep Graph Neural Nets from Suspended Animation," *arXiv:1909.05729*, 2019.
[141] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," in *NeurIPS*, 2016.
[142] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images," in *ECCV*, 2018.
[143] E. Smith, S. Fujimoto, A. Romero, and D. Meger, "GEOMetrics: Exploiting Geometric Structure for Graph-Encoded Objects," in *ICML*, 2019.
[144] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes," in *CVPR*, 2017.
[145] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," in *CVPR*, 2015.
[146] H. Gouraud, "Computer Display of Curved Surfaces," The University of Utah, Tech. Rep., 1971.
[147] A. Griewank and S. Reese, "On the calculation of Jacobian matrices by the Markowitz rule," 1991.
[148] T. Yoshida, "Derivation of a Computational Process for Partial Derivatives of Functions Using Transformations of a Graph," *Transactions of Information Processing Society of Japan*, vol. 28, no. 11, pp. 1112–1120, 1987.
[149] B. Segovia, J. C. Iehl, and B. Péroche, "Coherent Metropolis Light Transport with Multiple-Try Mutations," in *Soumis a Eurographics Symp. on Rendering*, 2007.
[150] B. Segovia, J.-C. Iehl, and B. Péroche, "Metropolis Instant Radiosity," *CGF*, vol. 26, no. 3, pp. 425–434, 2007.
[151] D. Blythe, "The direct3d 10 system," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 724–734, 2006.
[152] The Khronos Group Inc, "OpenGL Overview," https://www.khronos.org/opengl/.
[153] ——, "Vulkan Overview," https://www.khronos.org/vulkan/.
[154] "Unity Real-Time Development Platform," https://unity.com/.
[155] "Unreal Engine," https://www.unrealengine.com/.
[156] Autodesk, "Arnold Renderer," https://www.arnoldrenderer.com/.
[157] I. Georgiev, T. Ize, M. Farnsworth, R. Montoya-Vozmediano, A. King, B. V. Lommel, A. Jimenez, O. Anson, S. Ogaki, E. Johnston *et al.*, "Arnold: A brute-force production path tracer," *ACM Trans. Graph.*, vol. 37, no. 3, pp. 32:1–32:12, 2018.
[158] Chaos Group, "V-Ray," https://www.chaosgroup.com/.
[159] Pixar, "RenderMan," https://renderman.pixar.com/.
[160] S. Upstill, *The RenderMan Companion*. Addison-Wesley Professional, 1989.
[161] Autodesk, "Maya," https://www.autodesk.com/products/maya/.
[162] Autodesk, "3ds Max," https://www.autodesk.com/products/3ds-max/.
[163] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst, "Embree: A Kernel Framework for Efficient CPU Ray Tracing," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 143:1–143:8, 2014.
[164] I. Wald, G. P. Johnson, J. Amstutz, C. Brownlee, A. Knoll, J. Jeffers, J. Günther, and P. Navrátil, "Ospray-a cpu ray tracing framework for scientific visualization," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 931–940, 2016.
[165] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison *et al.*, "OptiX: A General Purpose Ray Tracing Engine," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 66:1–66:13, 2010.
[166] A. Keller, C. Wächter, M. Raab, D. Seibert, D. van Antwerpen, J. Korndörfer, and L. Kettner, "The iray light transport simulation and rendering system," in *ACM SIGGRAPH 2017 Talks*, 2017, pp. 1–2.

Hiroharu Kato received his BS degree (2012) in Engineering and his MS degree (2014) in Information Science and Technology from the University of Tokyo, Japan. After spending two and a half years at Sony Corporation as a research engineer, he is currently a researcher at Preferred Networks, Inc. He is also a Ph.D. student at the University of Tokyo since 2016.

Deniz Beker obtained his BSc. degree from Isk University in 2011 and his MSc. degree from Yeditepe University in 2014 both in Electrical and Electronics Engineering. He currently works at Preferred Networks as an engineer and engineering manager. His research interests are 3D computer vision and autonomous driving problems.

Mihai Morariu obtained a BSc. (Mathematics) degree from the University of Bucharest (Romania) in 2011 and a MSc. (Artificial Intelligence) from the Univesity of Amsterdam (The Netherlands) in 2013. After having worked for several years in The Netherlands as a computer vision engineer (in robotics), he relocated to Tokyo in 2018. He is now working as an engineer for Preferred Networks, Inc.

Takahiro Ando obtained his BSc. degree from Tsukuba University in 2002. He currently works at Preferred Networks as engineer. His research interests are 3D computer vision and computer graphics.

Toru Matsuoka obtained his MSc. degree in Computer Science from Takushoku University in 2004. He started his career as a CAD/CAM software developer in 2005. After that, he joined Preferred Networks as an engineer in 2017 after having worked for a video production company and a game company.

Wadim Kehl is an ML engineer at the Toyota Research Institute - Advanced Development (TRI-AD) in Tokyo, Japan. His research focuses on the interface between 3D object perception and differentiable visual representations, and how to apply these insights to practical problems. He received his BSc. from the University of Bonn, and his MSc. as well as PhD from the Technical University of Munich. He has authored over 20 patents and publications, presented at CVPR/ICCV/ECCV, and is a reviewer for most top-tier conferences and journals in Computer Vision and ML.

Adrien Gaidon is the Head of Machine Learning Research at the Toyota Research Institute (TRI) in Los Altos, CA, USA. Adrien's research focuses on scaling up ML for robot autonomy, spanning Scene and Behavior Understanding, Simulation for Deep Learning, 3D Computer Vision, and Self-Supervised Learning. He received his PhD from Microsoft Research - Inria Paris in 2012, has over 40 publications and patents in ML/CV/AI, top entries in international Computer Vision competitions, multiple best reviewer awards, international press coverage for his work on Deep Learning with simulation, was a guest editor for the International Journal of Computer Vision, and co-organized multiple workshops on Autonomous Driving at CVPR/ECCV/ICML.