Differentiable Surface Splatting for Point-based Geometry Processing

WANG YIFAN, ETH Zurich, Switzerland
FELICE SERENA, ETH Zurich, Switzerland
SHIHAO WU, ETH Zurich, Switzerland
CENGIZ ÖZTIRELI, Disney Research Zurich, Switzerland
OLGA SORKINE-HORNUNG, ETH Zurich, Switzerland

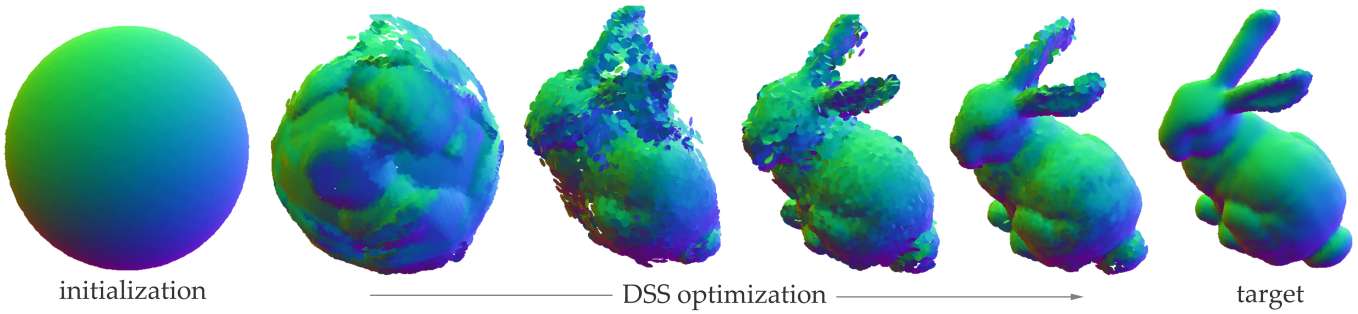


Fig. 1. Using our differentiable point-based renderer, scene content can be optimized to match target rendering. Here, the positions and normals of points are optimized in order to reproduce the reference rendering of the Stanford bunny. It successfully deforms a sphere to a target bunny model, capturing both large scale and fine-scale structures. From left to right are the input points, the results of iteration 18, 57, 198, 300, and the target.

We propose Differentiable Surface Splatting (DSS), a high-fidelity differentiable renderer for point clouds. Gradients for point locations and normals are carefully designed to handle discontinuities of the rendering function. Regularization terms are introduced to ensure uniform distribution of the points on the underlying surface. We demonstrate applications of DSS to inverse rendering for geometry synthesis and denoising, where large scale topological changes, as well as small scale detail modifications, are accurately and robustly handled without requiring explicit connectivity, outperforming state-of-the-art techniques. The data and code are at https://github.com/yifita/DSS.

CCS Concepts: • **Computing methodologies** → **Point-based models**; Computer vision; *Machine learning*; *Rendering*.

Additional Key Words and Phrases: differentiable renderer, neural renderer, deep learning

1 INTRODUCTION

Differentiable processing of scene-level information in the image formation process is emerging as a fundamental component for both 3D scene and 2D image and video modeling. The challenge of developing a differentiable renderer lies at the intersection of computer graphics, vision, and machine learning, and has recently attracted a lot of attention from all communities due to its potential to revolutionize digital visual data processing and high relevance for a wide range of applications, especially when combined with the contemporary neural network architectures [Kato et al. 2018; Liu et al. 2018; Loper and Black 2014; Petersen et al. 2019; Yao et al. 2018].

Authors' addresses: Wang Yifan, yifan.wang@inf.ethz.ch, ETH Zurich, Switzerland; Felice Serena, fserena@student.ethz.ch, ETH Zurich, Switzerland; Shihao Wu, shihao.wu@inf.ethz.ch, ETH Zurich, Switzerland; Cengiz Öztireli, cengiz.oztireli@disneyresearch.com, Disney Research Zurich, Switzerland; Olga Sorkine-Hornung, sorkine@inf.ethz.ch, ETH Zurich, Switzerland.

A differentiable renderer (DR) $\mathcal{R}$ takes scene-level information θ such as 3D scene geometry, lighting, material and camera position as input, and outputs a synthesized image $\mathbb{I} = \mathcal{R}(\theta)$. Any changes in the image $\mathbb{I}$ can thus be propagated to the parameters θ, allowing for image-based manipulation of the scene. Assuming a differentiable loss function $\mathcal{L}(\mathbb{I}) = \mathcal{L}(\mathcal{R}(\theta))$ on a rendered image $\mathbb{I}$, we can update the parameters θ with the gradient $\frac{\partial \mathcal{L}}{\partial \mathbb{I}} \frac{\partial \mathbb{I}}{\partial \theta}$. This view provides a generic and powerful shape-from-rendering framework where we can exploit vast image datasets available, deep learning architectures and computational frameworks, as well as pre-trained models. The challenge, however, is being able to compute the gradient $\frac{\partial \mathbb{I}}{\partial \theta}$ in the renderer.

Existing DR methods can be classified into three categories based on their geometric representation: voxel-based [Liu et al. 2017; Nguyen-Phuoc et al. 2018; Tulsiani et al. 2017], mesh-based [Kato et al. 2018; Liu et al. 2018; Loper and Black 2014], and point-based [Insafutdinov and Dosovitskiy 2018; Lin et al. 2018; Rajeswar et al. 2018; Roveri et al. 2018a]. Voxel-based methods work on volumetric data and thus come with high memory requirements even for relatively coarse geometries. Mesh-based DRs solve this problem by exploiting the sparseness of the underlying geometry in the 3D space. However, they are bound by the mesh structure with limited room for global and topological changes, as connectivity is not differentiable. Equally importantly, acquired 3D data typically comes in an unstructured representation that needs to be converted into a mesh form, which is itself a challenging and error-prone operation. *Point-based* DRs circumvent these problems by directly operating on point samples of the geometry, leading to flexible and efficient processing. However, existing point-based DRs use simple rasterization techniques such as forward-projection or depth maps, and thus come with well-known deficiencies in point cloud processing when

capturing fine geometric details, dealing with gaps and occlusions between near-by points, and forming a continuous surface.

In this paper, we introduce Differentiable Surface Splatting (DSS), the first *high fidelity point based differentiable renderer*. We utilize ideas from surface splatting [Zwicker et al. 2001], where each point is represented as a disk or ellipse in the object space, which is projected onto the screen space to form a splat. The splats are then interpolated to encourage hole-free and antialiased renderings. For inverse rendering, we carefully design gradients with respect to point locations and normals by taking each forward operation apart and utilizing domain knowledge. In particular, we introduce regularization terms for the gradients to carefully drive the algorithms towards the most plausible point configuration. There are infinitely many ways splats can form a given image due to the high degree of freedom of point locations and normals. Our inverse pass ensures that points stay on local geometric structures with uniform distribution.

We apply DSS to render multi-view color images as well as auxiliary maps from a given scene. We process the rendered images with state-of-the-art techniques and show that this leads to high-quality geometries when propagated utilizing DSS. Experiments show that DSS yields significantly better results compared to previous DR methods, especially for substantial topological changes and geometric detail preservation. We focus on the particularly important application of point cloud denoising. The implementation of DSS, as well as our experiments, will be available upon publication.

2 RELATED WORK

In this section we provide some background and review the state of the art in differentiable rendering and point based processing.

2.1 Differentiable rendering

We first discuss general DR frameworks, followed by DRs for specific purposes.

Loper and Black [2014] developped a differentiable renderer framework called OpenDR that approximates a primary renderer and computes the gradients via automatic differentiation. Neural mesh renderer (NMR) [Kato et al. 2018] approximates the backward gradient for the rasterization operation using a handcrafted function for visibility changes. Liu et al. [2018] propose Paparazzi, an analytic DR for mesh geometry processing using image filters. In concurrent work, Petersen et al. [2019] present *Pix2Vex*, a C^∞ differentiable renderer via soft blending schemes of nearby triangles, and Liu et al. [2019] introduce *Soft Rasterizer*, which renders and aggregates the probabilistic maps of mesh triangles, allowing flowing gradients from the rendered pixels to the occluded and far-range vertices. All these generic DR frameworks rely on mesh representation of the scene geometry. We summarize the properties of these renderers in Table 1 and discuss them in greater detail in Sec. 3.2.

Numerous recent works employed DR for learning based 3D vision tasks, such as single view image reconstruction [Pontes et al. 2017; Vogels et al. 2018; Yan et al. 2016; Zhu et al. 2017], face reconstruction [Richardson et al. 2017], shape completion [Hu et al. 2019], and image synthesis [Sitzmann et al. 2018]. To describe a few, Pix2Scene [Rajeswar et al. 2018] uses a point based DR to learn implicit 3D representations from images. However, Pix2Scene renders one surfel for each pixel and does not use screen space blending. Nguyen-Phuoc et al. [2018] and Insafutdinov and Dosovitskiy [2018] propose neural DRs using a volumetric shape representation, but the resolution is limited in practice. Li et al. [2018] and Azinović et al. [2019] introduce a differentiable ray tracer to implement the differentiability of physics based rendering effects, handling e.g. camera position, lighting and texture. While DSS could be extended and adapted to the above applications, in this paper, we demonstrate its power in shape editing, filtering, and reconstruction.

A number of works render depth maps of point sets [Insafutdinov and Dosovitskiy 2018; Lin et al. 2018; Roveri et al. 2018b] for point cloud classification or generation. These renderers do not define proper gradients for updating point positions or normals, thus they are commonly applied as an add-on layer behind a point processing network, to provide 2D supervision. Typically, their gradients are defined either only for depth values [Lin et al. 2018], or within a small local neighborhood around each point. Such gradients are not sufficient to alter the shape of a point cloud, as we show in a pseudo point renderer in Fig. 12.

The differentiable rendering also relates to shape-from-shading techniques [Langguth et al. 2016; Maier et al. 2017; Sengupta et al. 2018; Shi et al. 2017] that extract shading and albedo information for geometry processing and surface reconstruction. However, the framework proposed in this paper can be used seamlessly with contemporary deep neural networks, opening a variety of new applications.

2.2 Point-based geometry processing and rendering

With the proliferation of 3D scanners and depth cameras, the capture and processing of 3D point clouds is becoming commonplace. The noise, outliers, incompleteness and misalignments persisting in the raw data pose significant challenges for point cloud filtering, editing, and surface reconstruction [Berger et al. 2017].

Early optimization based point set processing methods rely on shape priors. Alexa and colleagues [2003] introduce the moving least squares (MLS) surface model, assuming a smooth underlying surface. Aiming to preserve sharp edges, Öztireli et al. [2009] propose the robust implicit moving least squares (RIMLS) surface model. Huang et al. [2013] employ an anisotropic *weighted locally optimal projection* (WLOP) operator [Huang et al. 2009; Lipman et al. 2007] and a progressive edge aware resampling (EAR) procedure to consolidate noisy input. Lu et al. [2018] formulate WLOP with a Gaussian mixture model and use point-to-plane distance for point set processing (GPF). These methods depend on the fitting of local geometry, e.g. normal estimation, and struggle with reconstructing multi-scale structures from noisy input.

Advanced learning-based methods for point set processing are currently emerging, encouraged by the success of deep learning. Based on PointNet [Qi et al. 2017], PCPNET [Guerrero et al. 2018] and PointCleanNet [Rakotosaona et al. 2019] estimate local shape properties from noisy and outlier-ridden point sets; EC-Net [Yu et al. 2018] learns point cloud consolidation and restoration of sharp features by minimizing a point-to-edge distance, but it requires edge

method	objective	position update	depth update	normal update	occlusion	silhouette change	topology change
OpenDR	mesh	✓	✗	via position change	✗	✓	✗
NMR	mesh	✓	✗	via position change	✗	✓	✗
Paparazzi	mesh	limited	limited	via position change	✗	✗	✗
Soft Rasterizer	mesh	✓	✓	via position change	✓	✓	✗
Pix2Vex	mesh	✓	✓	via position change	✓	✓	✗
Ours	points	✓	✓	✓	✓	✓	✓

Table 1. Comparison of generic differential renderers. By design, OpenDR [Loper and Black 2014] and NMR [Kato et al. 2018] do not propagate gradients to depth; Paparazzi [Liu et al. 2018] has limitation in updating the vertex positions in directions orthogonal their face normals, thus can not alter the silhouette of shapes; Soft Rasterizer [Liu et al. 2019] and Pix2Vex [Petersen et al. 2019] can pass gradient to occluded vertices, through blurred edges and transparent faces. All mesh renderers do not consider the normal field directly and cannot modify mesh topology. Our method uses a point cloud representation, updates point position and normals jointly, considers the occluded points and visibility changes and enables large deformation including topology changes.

annotation for the training data. Hermosilla et al. [2019] propose an unsupervised point cloud cleaning method based on Monte Carlo convolution [Hermosilla et al. 2018]. Roveri et al. [2018a] present a projection based differentiable point renderer to convert unordered 3D points to 2D height maps, enabling the use of convolutional layers for height map denoising before back-projecting the smoothed pixels to the 3D point cloud. In contrast to the commonly used Chamfer or EMD loss [Fan et al. 2017], our DSS framework, when used as a loss function, is compatible with convolutional layers and is sensitive to the exact point distribution pattern.

Surface splatting is fundamental to our method. Splatting has been developed for simple and efficient point set rendering and processing in the early seminal point based works [Pfister et al. 2000; Zwicker et al. 2002, 2001, 2004]. Recently, point based techniques have gained much attention for their superior potential in geometric learning. To the best of our knowledge, we are the first to implement high-fidelity differentiable surface splatting.

3 METHOD

In essence, a differentiable renderer $\mathcal{R}$ is designed to propagate image-level changes to scene-level parameters θ. This information can be used to optimize the parameters so that the rendered image $\mathbb{I} = \mathcal{R}(\theta)$ matches a reference image $\mathbb{I}^*$. Typically, θ includes the coordinates, normals and colors of the points, camera position and orientation, as well as lighting. Formally, this can be formulated as an optimization problem

$$\theta^* = \arg\min_{\theta} \mathcal{L}\left(\mathcal{R}(\theta), \mathbb{I}^*\right), \tag{1}$$

where $\mathcal{L}$ is the image loss, measuring the distance between the rendered and reference images.

Methods to solve the optimization problem (1) are commonly based on gradient descent which requires $\mathcal{R}$ to be differentiable with respect to θ. However, gradients w.r.t. point coordinates $\mathbf{p}$ and normals $\mathbf{n}$, i.e., $\frac{d\mathbb{I}}{d\mathbf{p}}$ and $\frac{d\mathbb{I}}{d\mathbf{n}}$, are not defined everywhere, since $\mathcal{R}$ is a discontinuous function due to occlusion events and edges.

The key to our method is two-fold. First, we define a gradient $\frac{d\mathbb{I}}{d\mathbf{p}}$ and $\frac{d\mathbb{I}}{d\mathbf{n}}$ which enables information propagation from long-range pixels without additional hyper-parameters. Second, to address the optimization difficulty that arises from the significant number of degrees of freedom due to the unstructured nature of points, we introduce regularization terms that contribute to obtaining clean and smooth surface points.

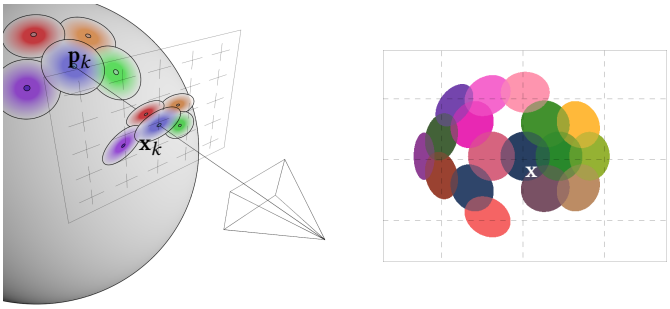


Fig. 2. Illustration of forward splatting using EWA [Zwicker et al. 2001]. A point in space $\mathbf{p}_k$ is rendered as an anisotropic ellipse centered at the projection point $\mathbf{x}_k$. The final pixel value $\mathbb{I}_\mathbf{x}$ at a pixel $\mathbf{x}$ in the image (shown on the right) is the normalized sum of all such ellipses overlapping at $\mathbf{x}$.

In this section, we first review screen space EWA (elliptical weighted average) [Heckbert 1989; Zwicker et al. 2001], which we adopt to efficiently render high-quality realistic images from point clouds. Then we propose an occlusion-aware gradient definition for the rasterization step, which, unlike previously proposed differential mesh renderers, propagates gradients to depth and allows large deformation. Lastly, we introduce two novel regularization terms for generating clean surface points.

3.1 Forward pass

The forward pass refers to the generation of a 2D image from 3D scene-level information, $\mathbb{I} = \mathcal{R}(\theta)$. Our forward pass closely follows the screen space elliptical weighted average (EWA) filtering described in [Zwicker et al. 2001]. In the following, we briefly review the derivation of EWA filters.

In a nutshell, the idea of screen space EWA is to apply an isotropic Gaussian filter to the attribute Φ of a point in the tangent plane (defined by the normal at that point). The projection onto the image plane defines elliptical Gaussians, which, after truncation to bounded support, form a disk, or splat, as shown in Fig. 2. For a point $\mathbf{p}_k$, we write the filter weight of the isotropic Gaussian at

position $\mathbf{p}$ as

$$\mathcal{G}_{\mathbf{p}_k,\mathbf{V}_k}(\mathbf{p}) = \frac{1}{2\pi|\mathbf{V}_k|^{\frac{1}{2}}} e^{(\mathbf{p}-\mathbf{p}_k)^\top \mathbf{V}_k^{-1}(\mathbf{p}-\mathbf{p}_k)}, \quad \mathbf{V}_k = \sigma_k^2\mathbf{I}, \tag{2}$$

where σ_k is the standard deviation and $\mathbf{I}$ is the identity matrix.

Now we consider the projected Gaussian in screen space. Points $\mathbf{p}_k$ and $\mathbf{p}$ are projected to $\mathbf{x}_k$ and $\mathbf{x}$, respectively. We write the Jacobian of this projection from the tangent plane to the image plane as $\mathbf{J}_k$; we refer the reader to the original surface splatting paper [Zwicker et al. 2001] for the derivation of $\mathbf{J}_k$. Then at $\mathbf{x}$, the screen space elliptical Gaussian weight is

$$\begin{aligned} r_k(\mathbf{x}) &= \mathcal{G}_{\mathbf{V}_k}\left(\mathbf{J}_k^{-1}(\mathbf{x}-\mathbf{x}_k)\right) \\ &= \frac{1}{\left|\mathbf{J}_k^{-1}\right|}\mathcal{G}_{\mathbf{J}_k\mathbf{V}_k\mathbf{J}_k^\top}(\mathbf{x}-\mathbf{x}_k). \end{aligned} \tag{3}$$

Note that r_k is determined by the point position $\mathbf{p}_k$ and the normal $\mathbf{n}_k$, because $\mathbf{J}_k$ is determined by $\mathbf{p}_k$ and $\mathbf{n}_k$.

Next, a low-pass Gaussian filter with variance $\mathbf{I}$ is convolved with Eq. (3) in screen space. Thus the final elliptical Gaussian is

$$\bar{\rho}_k(\mathbf{x}) = \frac{1}{\left|\mathbf{J}_k^{-1}\right|}\mathcal{G}_{\mathbf{J}_k\mathbf{V}_k\mathbf{J}_k^\top+\mathbf{I}}(\mathbf{x}-\mathbf{x}_k). \tag{4}$$

In the final step, two sources of discontinuity are introduced to the fully differentiable $\bar{\rho}$. First, for computational reasons, we limit the elliptical Gaussians to a limited support in the image plane for all $\mathbf{x}$ outside a cutoff radius C, i.e., $\frac{1}{2}\mathbf{x}^\top\left(\mathbf{J}\mathbf{V}_k\mathbf{J}^\top+\mathbf{I}\right)\mathbf{x} > C$. Second, we set the Gaussian weights for occluded points to zero. Specifically, we keep a list of the maximum K (we choose $K=5$) closest points at each pixel position, and compute their depth difference to the front-most point, and then set the Gaussian weights to zero for points that are behind the front-most point by more than a threshold $\mathcal{T}$ (we set $\mathcal{T}$ = 1% of the bounding box diagonal length). These K points are cached for gradient evaluation in backward pass, as will be explained in Sec. 3.2.

The resulting truncated Gaussian weight, denoted as ρ_k, can be formally defined as

$$\rho_k(\mathbf{x}) = \begin{cases} 0, & \text{if } \frac{1}{2}\mathbf{x}^\top\left(\mathbf{J}\mathbf{V}_k\mathbf{J}^\top+\mathbf{I}\right)\mathbf{x} > C, \\ 0, & \text{if } \mathbf{p}_k \text{ is occluded}, \\ \bar{\rho}_k, & \text{otherwise}. \end{cases} \tag{5}$$

The final pixel value is simply the normalized sum of all filtered point attributes $\{\mathbf{w}_k\}_{k=0}^N$ evaluated at the center of pixels, i.e.,

$$\mathbb{I}_\mathbf{x} = \frac{\sum_{k=0}^{N-1}\rho_k(\mathbf{x})\,\mathbf{w}_k}{\sum_{k=0}^{N-1}\rho_k(\mathbf{x})}. \tag{6}$$

In practice, this summation can be greatly optimized by computing the bounding box of each ellipse and only considering points whose elliptical support covers the pixel $\mathbf{x}$.

The point value Φ can be any point attribute, e.g., albedo color, shading, depth value, normal vector, etc. In most of our experiments, we use diffuse shading under three orthogonally positioned RGB-colored sun lights. This way, Φ carries strong information about point normals, and at the same time it is independent of point position (unlike with point lights), which greatly simplifies the factorization for gradient computation, as explained in Sec. 3.2.

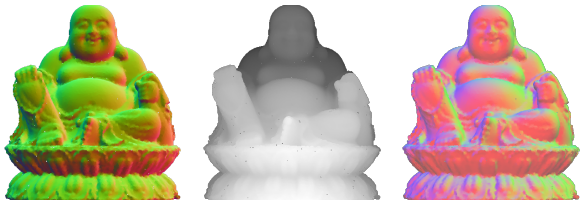


Fig. 3. Examples of images rendered using DSS. From left to right, we render the normals, inverse depth values and diffuse shading with three RGB-colored sun light sources.

Fig. 3 shows some examples of rendered images. Unlike many pseudo renderers which achieve differentiability by blurring edges and transparent surfaces, our rendered images faithfully depict the actual geometry in the scene.

3.2 Backward pass

The backward pass refers to the information flow from the rendered image $\mathbb{I} = \mathcal{R}(\theta)$ to the scene parameters θ based on approximating the gradient $\frac{d\mathbb{I}}{d\theta}$. As discussed, the key to address the discontinuity of $\mathcal{R}$ lies in the approximation of the gradient $\frac{d\mathbb{I}}{d\mathbf{p}}$ and $\frac{d\mathbb{I}}{d\mathbf{n}}$.

The discontinuity of $\mathcal{R}$ is encapsulated in the truncated Gaussian weights ρ_k as described Eq. (5). We can factorize the discontinuous ρ_k into the fully differentiable term $\bar{\rho}_k$ and a discontinuous visibility term $h_\mathbf{x} \in \{0,1\}$, i.e., $\rho_k = h_\mathbf{x}\bar{\rho}_k$, where $h_\mathbf{x}$ is defined as

$$h_\mathbf{x}(\mathbf{p}_k) = \begin{cases} 0, & \text{if } \frac{1}{2}\mathbf{x}^\top\left(\mathbf{J}\mathbf{V}_k\mathbf{J}^\top+\mathbf{I}\right)\mathbf{x} > C, \\ 0, & \text{if } \mathbf{p}_k \text{ is occluded}, \\ 1, & \text{otherwise}. \end{cases} \tag{7}$$

Note that even though $h_\mathbf{x}$ is indirectly influenced by $\mathbf{n}_k$ through $\mathbf{J}$, since this only impacts the visibility of a small set of pixels around the ellipse, we omit this $\mathbf{n}_k$ in this formulation. Therefore, if we write $\mathbb{I}_\mathbf{x}$ as a function of $\mathbf{w}_k$, $\bar{\rho}_k$ and h_k, then by the chain rule we have

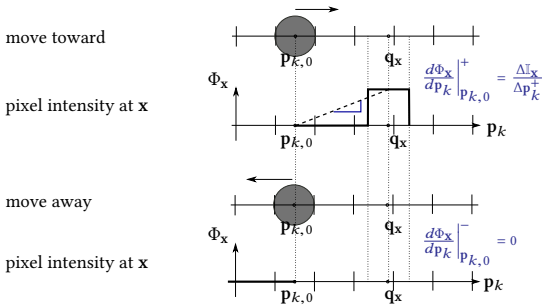
$$\frac{d\mathbb{I}_\mathbf{x}(\mathbf{w}_k,\bar{\rho}_k,h_\mathbf{x})}{d\mathbf{n}_k} = \frac{\partial\mathbb{I}_\mathbf{x}}{\partial\mathbf{w}_k}\frac{\partial\mathbf{w}_k}{\partial\mathbf{n}_k} + \frac{\partial\mathbb{I}_\mathbf{x}}{\partial\bar{\rho}_k}\frac{\partial\bar{\rho}_k}{\partial\mathbf{n}_k}, \tag{8}$$

$$\frac{d\mathbb{I}_\mathbf{x}(\mathbf{w}_k,\bar{\rho}_k,h_\mathbf{x})}{d\mathbf{p}_k} = \frac{\partial\mathbb{I}_\mathbf{x}}{\partial\mathbf{w}_k}\frac{\partial\mathbf{w}_k}{\partial\mathbf{p}_k} + \frac{\partial\mathbb{I}_\mathbf{x}}{\partial\bar{\rho}_k}\frac{\partial\bar{\rho}_k}{\partial\mathbf{p}_k} + \frac{\partial\mathbb{I}_\mathbf{x}}{\partial h_\mathbf{x}}\frac{\partial h_\mathbf{x}}{\partial\mathbf{p}_k}, \tag{9}$$

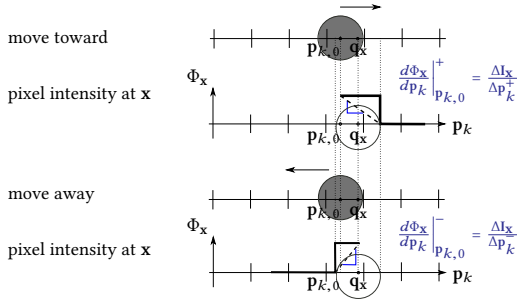
where Eq. (8) is fully differentiable but Eq. (9) is not, as $\frac{\partial h_\mathbf{x}}{\partial\mathbf{p}_k}$ is undefined at the edge of ellipses.

We focus on the partial gradient $\frac{\partial\mathbb{I}_\mathbf{x}}{\partial h_\mathbf{x}}\frac{\partial h_\mathbf{x}}{\partial\mathbf{p}_k}$. Denoting $\Phi_\mathbf{x}(\mathbf{p}_k) = \mathbb{I}_\mathbf{x}(h_\mathbf{x}(\mathbf{p}_k))$, this gradient can be written as $\frac{d\Phi_\mathbf{x}}{d\mathbf{p}_k}$, which describes the change of a pixel intensity $\mathbb{I}_\mathbf{x}$ due to the visibility change of a point caused by its varying position $\mathbf{p}_k$.

1D scenario. Let us first consider a simplified scenario where a single point only moves in 1D space. As shown in Fig. 4, $\Phi_\mathbf{x}$ is generally discontinuous; it is zero almost everywhere except in a small region around $\mathbf{q}_\mathbf{x}$, the coordinates of the pixel $\mathbf{x}$ back-projected to world coordinates. Similar to NMR [Kato et al. 2018],



(a) The ellipse centered at $\mathbf{p}_{k,0}$ is not visible at $\mathbf{x}$.



(b) The ellipse centered at $\mathbf{p}_{k,0}$ is visible at $\mathbf{x}$.

Fig. 4. An illustration of the artificial gradient in two 1D scenarios: the ellipse centered at $\mathbf{p}_{k,0}$ is invisible (Fig. 4a) and visible (Fig. 4b) at pixel $\mathbf{x}$. $\Phi_{\mathbf{x},k}$ is the pixel intensity $\mathbb{I}_\mathbf{x}$ as a function of point position $\mathbf{p}_k$, $\mathbf{q}_\mathbf{x}$ is the coordinates of the pixel $\mathbf{x}$ back-projected to world coordinates. Notice the ellipse has constant pixel intensity after normalization (Eq. (6)). We approximate the discontinuous $\Phi_{\mathbf{x},k}$ as a linear function defined by the change of pixel intensity $\Delta\mathbb{I}_\mathbf{x}$ and the movement of the $\Delta\mathbf{p}_k$ during a visibility switch. As $\mathbf{p}_k$ moves toward ($\Delta\mathbf{p}^+_k$) or away ($\Delta\mathbf{p}^-_k$) from the pixel, we obtain two different gradient values. We define the final gradient as their sum.

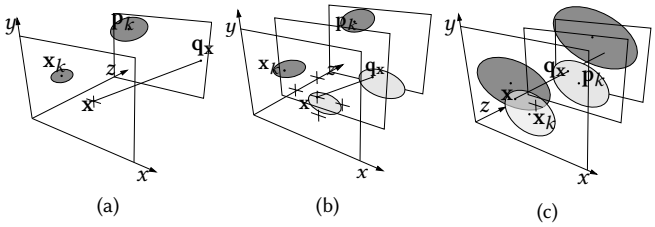


Fig. 5. Illustration of the 3 cases for evaluating Eq. (10) for 3D point clouds.

we approximate $\Phi_\mathbf{x}$ as a linear function defined by the change of point position $\Delta\mathbf{p}_k$ and the pixel intensity $\Delta\mathbb{I}$ before and after the visibility change.

As $\mathbf{p}_k$ moves toward or away from $\mathbf{q}_\mathbf{x}$, we obtain two different linear functions with gradients $\frac{d\Phi_\mathbf{x}}{d\mathbf{p}_k}\Big|^{+}_{\mathbf{p}_{k,0}}$ and $\frac{d\Phi_\mathbf{x}}{d\mathbf{p}_k}\Big|^{-}_{\mathbf{p}_{k,0}}$, respectively. Specifically, when $\mathbf{p}_k$ is invisible at $\mathbf{x}$ (Fig. 4a), moving away will always induce zero gradient, while when $\mathbf{p}_k$ is visible, we obtain two gradients with opposite signs (Fig. 4b). The final gradient is defined as the sum of both, i.e.,

$$\frac{d\Phi_x}{d\mathbf{p}_k}\Big|_{\mathbf{p}_{k,0}} = \begin{cases} \frac{\Delta\mathbb{I}_\mathbf{x}}{\|\Delta\mathbf{p}^+_k\|^2+\epsilon}\Delta\mathbf{p}^+_k, & \mathbf{p}_k \text{ invisible at } \mathbf{x} \\ \frac{\Delta\mathbb{I}_\mathbf{x}}{\|\Delta\mathbf{p}^-_k\|^2+\epsilon}\Delta\mathbf{p}^-_k + \frac{\Delta\mathbb{I}_\mathbf{x}}{\|\Delta\mathbf{p}^+_k\|+\epsilon}\Delta\mathbf{p}^+_k, & \text{otherwise,} \end{cases} \tag{10}$$

where $\Delta\mathbf{p}^-_k$ and $\Delta\mathbf{p}^+_k$ denote the point movement toward and away from $\mathbf{x}$, starting from the current position $\mathbf{p}_{k,0}$. The value ϵ is a small constant (we set $\epsilon = 10^{-5}$). It prevents the gradient from becoming extremely large when $\mathbf{p}_k$ is close $\mathbf{q}_\mathbf{x}$, which would lead to overshooting, oscillation and other convergence problems.

3D cases. Extending the single point 1D-scenario to a point cloud in 3D requires evaluating $\Delta\mathbb{I}$ and $\Delta\mathbf{p}$ with care. As depicted in Fig. 5, the following cases are considered: (a) $\mathbf{p}_k$ is not visible at $\mathbf{x}$ and $\mathbf{x}$ is not rendered by any other ellipses *in front of* $\mathbf{p}_k$; (b) $\mathbf{p}_k$ is not visible at $\mathbf{x}$ and $\mathbf{x}$ is rendered by other ellipses *in front of* $\mathbf{p}_k$; (c) $\mathbf{p}_k$ is visible at $\mathbf{x}$.

For (a) and (c), we only need to compute the gradient in screen space, whereas for (b), $\mathbf{p}_k$ must move forward in order to become visible, resulting in a negative depth gradient. Furthermore, for (a) and (b) we evaluate the new $\mathbb{I}_\mathbf{x}$ using Eq. (6), *adding* the contribution from $\mathbf{p}_k$, while for (c) we need to *subtract* the contribution of $\mathbf{p}_k$, which may include previously occluded ellipses into Eq. (6). For this purpose, as mentioned in 3.1, we cache an ordered list of the top-K (we choose K=5) closest ellipses that can be projected onto each pixel and save their ρ, Φ and depth values during the forward pass. The value of K is related to the merging threshold $\mathcal{T}$, and as $\mathcal{T}$ is typically small, we find $K = 5$ is sufficient even for dense point clouds.

Finally, similar to NMR, when evaluating Eq. (10) for the optimization problem (1), we set the gradient to zero if the change of pixel intensity cannot reduce the image loss $\mathcal{L}$, i.e.,

$$\frac{d\Phi_x}{d\mathbf{p}_k}\Big|_{\mathbf{p}_k=\mathbf{p}_{k,0}} = 0 \quad \text{if} \quad \frac{d\mathcal{L}}{d\mathbb{I}_\mathbf{x}}\Delta\mathbf{I}_\mathbf{x} >= 0. \tag{11}$$

Comparison to other differentiable renderers. A few differential renderers have been proposed for meshes. In Paparazzi [Liu et al. 2018], the rendering function is simplified enough such that the gradients can be computed analytically, which is prohibitive for silhouette change where handling significant occlusion events is required. OpenDR [Loper and Black 2014] computes gradients only in screen space from a small set of pixels near the boundary, which is conceptually less accurate than our definition. SoftRasterizer [Liu et al. 2019] alters the forward rendering to make the rasterization step inherently differentiable; this leads to impeded rendering quality and relies on hyper-parameters to control the differentiability (i.e., support of non-zero gradient). The work related most closely to our approach in terms of gradient definition is the neural mesh renderer (NMR) [Kato et al. 2018]. We both construct $\Phi_\mathbf{x}$ depending on the change of pixel $\Delta\mathbb{I}_\mathbf{x}$, but our method differs from NMR in the following aspects: (1) we consider the movement of $\mathbf{p}_k$ in 3D space, while NMR only considers movement in the image plane, hence neglecting the gradient in z-dimension. (2) we define the

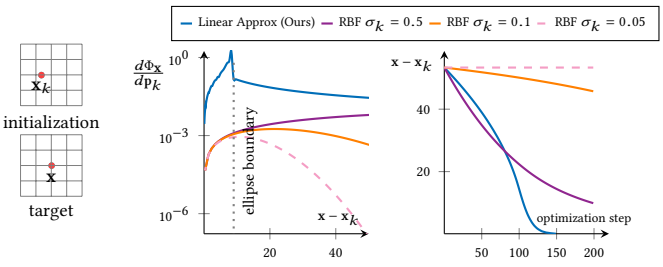


Fig. 6. Comparison between RBF-based gradient and our gradient approximation in terms of the gradient value at pixel $\mathbf{x}$ and residual in image space $\mathbf{x} - \mathbf{x}_k$ as we optimize the point position $\mathbf{p}_k$ in the initial rendered image to match the target image. While our approximation (blue) is invariant under the choice of the hyper-parameter σ_k, the RBF-based gradient (purple, orange and the dashed pink curves) is highly sensitive to its value. Small variations of σ_k can severely impact the convergence rate.

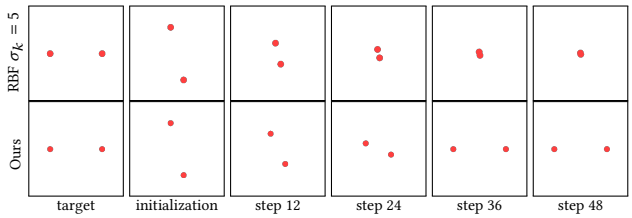


Fig. 7. Optimization progress using our gradient approximation and RBF-derived gradient. The RBF-derived gradient is prone to local minima when optimizing for multiple points.

gradient for all dimensions of $\mathbf{p}$ jointly. In contrast, NMR evaluates the 1D gradients separately and consequently considers only pixels in the same row and column; (3) we consider a set of occluded and occluding ellipses projected to pixel $\mathbf{x}$. This not only leads to more accurate gradient values, but also encourages noisy points inside the model to move onto the surface, to a position with matching pixel color.

Comparison to filter-based gradient approximation. Alternatively, related to SoftRasterizer [Liu et al. 2019] and Pix2Vex [Petersen et al. 2019], one can define the gradient of the discontinuous function $\Phi_{\mathbf{x},k}$ by replacing it with a C^∞ function, e.g., a radial basis function (RBF). This is a seemingly natural choice for EWA-based point rendering, since each point is represented as a RBF in the forward pass. We compare the RBF-derived gradient with our approximation in a single point 1D scenario, and evaluate the gradient value and convergence rate. As shown in Fig. 6, the RBF-derived gradient is highly sensitive to the Gaussian filter's standard deviation σ_k. A small σ_k leads to diminishing gradient for distant pixels, causing convergence issues, as demonstrated with the dashed plot. For a large σ_k, $\|\frac{d\Phi_\mathbf{x}}{d\mathbf{p}_k}\|$ can increase with $\mathbf{x} - \mathbf{x}_k$ when the pixel is outside the ellipse boundary; as a result, the optimization is prone to fall into a local minima in multi-point scenario as shown in Fig. 7. Lastly, it is not obvious how to extend the RBF derivation for the depth dimension, while the linear approximation naturally applies to all dimensions.

3.3 Surface regularization

The lack of structure in point clouds, while providing freedom of massive topology changes, can pose a significant challenge for optimization. First, the gradient derivation is entirely paralleled; as a result, points move irrespective of each other. Secondly, as the movement of points will only induce small and sparse changes in the rendered image, gradients on each point are less structured compared to corresponding gradients for meshes. Without proper regularization, one can quickly end up in local minima.

We propose regularization to address this problem based on two parts: a repulsion and a projection term. The repulsion term is aimed at generating uniform point distributions by maximizing the distances between its neighbors on a local projection plane, while the projection term preserves clean surfaces by minimizing the distance from the point to the surface tangent plane.

Both terms require finding a reliable surface tangent plane. However, this can be challenging, since during optimization, especially in the case of multi-view joint optimization, intermediate point clouds can be very noisy and contain many occluded points inside the model, hence we propose a weighted PCA to penalize the occluded inner points. In addition to the commonly used bilateral weights which considers both the point-to-point euclidean distance and the normal similarity, we propose a visibility weight, which penalizes occluded points, since they are more likely to be outliers inside the model.

Let $\mathbf{p}_i$ denote a point in question and $\mathbf{p}_k$ denote one point in its neighborhood, $\mathbf{p}_k \in \{\mathbf{p} | \ \|\mathbf{p} - \mathbf{p}_i\| \leq \mathcal{D}\}$, we propose computing a weighted PCA using the following weights

$$\psi_{ik} = \exp\left(-\frac{\|\mathbf{p}_i - \mathbf{p}_k\|^2}{\mathcal{D}^2}\right) \tag{12}$$

$$\theta_{ik} = \exp\left(-\frac{\left(1 - \mathbf{n}_k^\top \mathbf{n}_i\right)^2}{\max\left(1e^{-5}, 1 - \cos\left(\Theta\right)\right)^2}\right) \tag{13}$$

$$\phi_{ik} = \frac{1}{o_k + 1}, \tag{14}$$

where ψ_{ik} and θ_{ik} are bilateral weights which favor neighboring points that are spatially close and have similar normal orientation respectively, and ϕ_{ik} is the proposed visibility weight which is defined using an occlusion counter o_k that counts the number of times $\mathbf{p}_k$ is occluded in all camera views. Then a reliable projection plane can be obtained using singular value decomposition from weighted vectors $w_{ik}\left(\mathbf{p}_i - \sum_{k=0}^K w_{ik}\mathbf{p}_k\right)$, where $w_{ik} = \frac{\psi_{ik}\theta_{ik}\phi_{ik}}{\sum_{i=0}^K \psi_{ik}\theta_{ik}\phi_{ik}}$.

For the repulsion term, the projected point-to-point distance is obtained via $d_{ik} = \tilde{\mathbf{V}}\tilde{\mathbf{V}}^\top(\mathbf{p}_i - \mathbf{p}_k)$, where $\tilde{\mathbf{V}}$ contains the first 2 principal components. We define the repulsion loss as follows and minimize it together with the per-pixel image loss

$$\mathcal{L}_r = \frac{1}{N}\sum_N \sum_K \frac{\psi_{ik}}{d_{ik}^2 + 10^{-4}}. \tag{15}$$

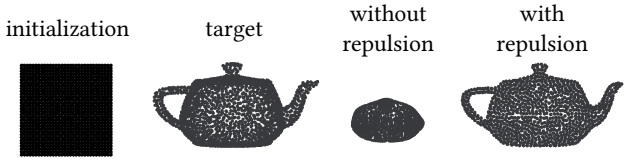


Fig. 8. The effect of repulsion regularization. We deform a 2D grid to the teapot. Without the repulsion term, points cluster in the center of the target shape. The repulsion term penalizes this type of local minima and encourages a uniform point distribution.

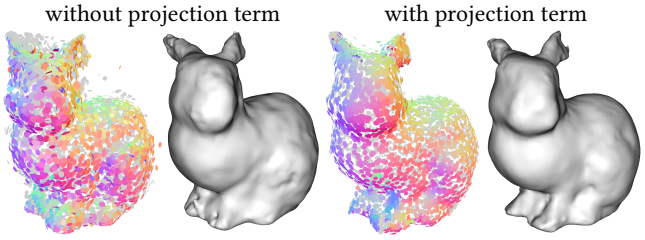


Fig. 9. The effect of projection regularization. The projection term effectively enforces points to form a local manifold. For a better visualization of outliers inside and outside of the object, we use a small disk radius and render the backside of the disks using light gray color.

For the projection term, we minimize the point-to-plane distance via $d_{ik} = \mathbf{V}_n \mathbf{V}^\top (\mathbf{p}_i - \mathbf{p}_k)$, where $\mathbf{V}_n$ is the last components. Correspondingly, the projection loss is defined as

$$\mathcal{L}_p = \frac{1}{N} \sum_N \sum_K w_{ik} d_{ik}^2. \tag{16}$$

The effect of repulsion and projection terms are clearly demonstrated in Fig. 8 and Fig. 9. In Fig. 8, we aim to move points lying on a 2D grid to match the silhouette of a 3D teapot. Without the repulsion term, points quickly shrink to the center of the reference shape, which is a common local minima since the gradient coming from surrounding pixels cancel each other out. With the repulsion term, the points can escape such local minima and distribute evenly inside the silhouette. In Fig. 9 we deform a sphere to bunny from 12 views. Without projection regularization, points are scattered within and outside the surface. In contrast, when the projection term is applied, we can obtain a clean and smooth surface.

4 IMPLEMENTATION DETAILS.

4.1 Optimization objective

We choose Symmetric Mean Absolute Percentage Error (SMAPE) as the image loss $\mathcal{L}_\mathbb{I}$. SMAPE is designed for high dynamic range images such as rendered images therefore it behaves more stable for unbounded values [Vogels et al. 2018]. It is defined as

$$\mathcal{L}_\mathbb{I} = \frac{1}{HW} \sum_{\mathbf{x} \in \mathbb{I}} \sum_c^C \frac{|\mathbb{I}_{\mathbf{x},c} - \mathbb{I}^*_{\mathbf{x},c}|}{|\mathbb{I}_{\mathbf{x},c}| + |\mathbb{I}^*_{\mathbf{x},c}| + \epsilon}, \tag{17}$$

where H and W are the dimensions of the image, the value of ϵ is typically chosen as 10^{-5}.

The total optimization objective corresponding to Eq. (1) for a set of views V amounts to

$$\sum_{v=0}^{V} \mathcal{L}\left(\mathbb{I}_v, \mathbb{I}^*_v\right) = \sum_{v=0}^{V} \mathcal{L}_\mathbb{I}\left(\mathbb{I}_v, \mathbb{I}^*_v\right) + \gamma_p \mathcal{L}_p + \gamma_r \mathcal{L}_r. \tag{18}$$

Loss weights γ_p and γ_r are typically chosen to be $0.02, 0.05$ respectively.

4.2 Alternating normal and point update

For meshes, the face normals are determined by point positions. For points, though, normals and point positions can be treated as independent entities thus optimized individually. Our pixel value factorization in Eq. (9) and Eq. (8) means that, the gradient on point positions $\mathbf{p}$ mainly stems from the visibility term, while gradients on normals $\mathbf{n}$ can be derived from $\mathbf{w}_k$ and ρ_k. Because the gradient w.r.t. $\mathbf{n}$ and $\mathbf{p}$ assumes the other stays fixed, we apply the update of $\mathbf{n}$ and $\mathbf{p}$ in an alternating fashion. Specifically, we start with normals, execute optimization for $T_\mathbf{n}$ times then we optimize point positions for $T_\mathbf{p}$ times.

As observed in many point denoising works [Guerrero et al. 2018; Huang et al. 2009; Öztireli et al. 2009], finding the right normal is the key for obtaining clean surfaces. Hence we efficiently utilize the improved normals even if the point positions are not being updated, in that we directly update the point positions using the gradient from the regularization terms $\frac{\partial \mathcal{L}_p}{\partial \mathbf{p}_k}$ and $\frac{\partial \mathcal{L}_r}{\partial \mathbf{p}_k}$. In fact, for local shape surface modification, this simple strategy consistently yields satisfying results.

4.3 Error-aware view sampling

In our experiments, we cover all possible angles by sampling camera positions from a hulling sphere using farthest point sampling. Then we randomly perturb the sampled position and set the camera to look at the center of the object. The sampling process is repeated periodically to further improve optimization.

However, for shapes with complex topology, such a sampling scheme is not enough. We propose an error-aware view sampling scheme which chooses the new camera positions based on the current image loss.

Specifically, we downsample the reference image and the rendered result, then compute the pixel position with the largest image error. Then we find K points whose projection is closest to the found pixel. The mean 3D position of these points will be the center of focus. Finally, we sample camera positions on a sphere around this focal point with a relatively small distance. Such techniques help us to improve point positions in small holes during large shape deformation.

5 RESULTS

We evaluate the performance of DSS by comparing it to state-of-the-art DRs, and demonstrate its applications in point-based geometry editing and filtering.

Our method is implemented in Pytorch [Paszke et al. 2017], we use stochastic gradient descent with Nesterov momentum [Sutskever et al. 2013] for optimization. A learning rate of 5 and 5000 is used for points and normals, respectively. In all experiments, we render

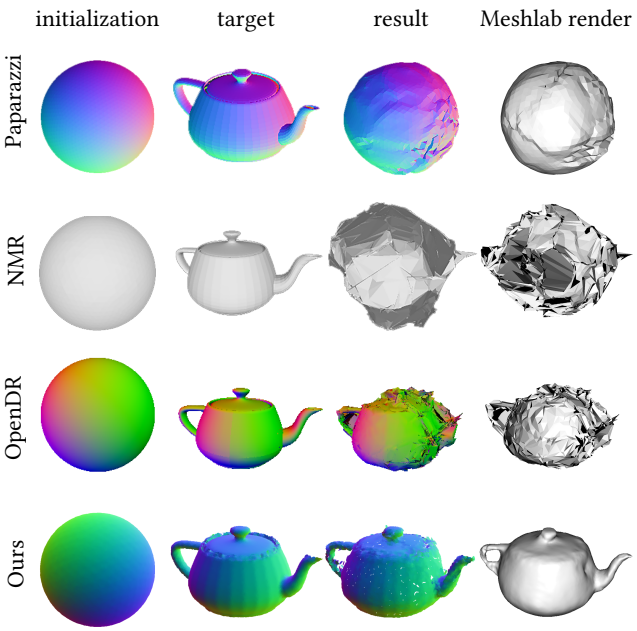


Fig. 10. Large shape deformation with topological changes, compared with three mesh-based DRs, namely Paparazzi [Liu et al. 2018], OpenDR [Loper and Black 2014] and Neural Mesh Renderer [Kato et al. 2018]. Compared to the mesh-based approaches, DSS faithfully recovers the handle and cover of the teapot thanks to the flexibility of the point-based representation.

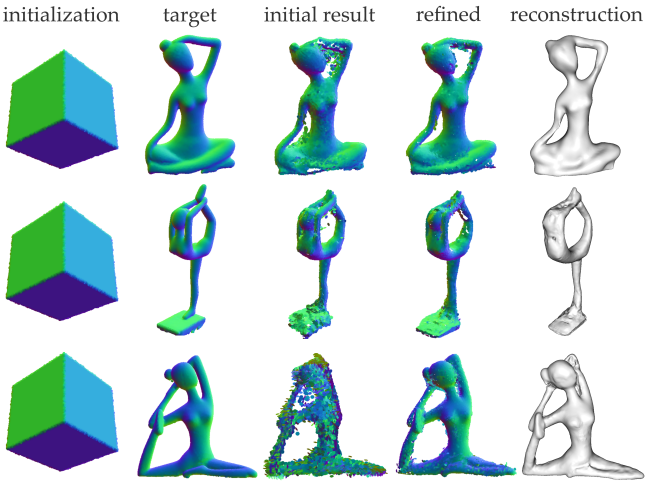


Fig. 11. DSS deforms a cube to three different Yoga models. Noisy points may occur when camera views are under-sampled or occluded (as shown in the initial result). We apply an additional refinement step improving the view sampling as described in Sec. 4.3.

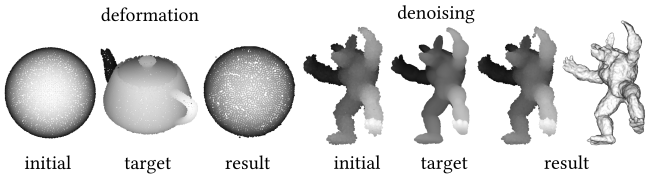


Fig. 12. A simple projection-based point renderer which renders depth values fails in deformation and denoising tasks.

in back-face culling mode with 256×256 resolution and diffuse shading, using RGB sun lights fixed relative to the camera position.

Unless otherwise stated, we optimize for up to 16 cycles of $T_{\mathbf{n}}$ and $T_{\mathbf{p}}$ optimization steps for point normal and position (for large deformation $T_{\mathbf{p}} = 25$ and $T_{\mathbf{n}} = 15$; for local surface editing $T_{\mathbf{n}} = 19$ and $T_{\mathbf{p}} = 1$). In each cycle, 12 randomly sampled views are used simultaneously for an optimization step. To test our algorithms for noise resilience, we use random white Gaussian noise with a standard deviation measured relative to the diagonal length of the bounding box of the input model. We refer to Appendix A for a detailed discussion of parameter settings.

5.1 Comparison of different DRs.

We compare DSS in terms of large geometry deformation to the state-of-the-art mesh-based DRs, i.e., OpenDR [Loper and Black 2014], NMR [Kato et al. 2018] and Paparazzi [Liu et al. 2018]. For the mesh DRs, we use the publicly available code provided by the authors and report the best results among experiments using different parameters (e.g., number of cameras and learning rate). All methods use the same initial and target shape, and similar camera positions.

Among the mesh-based methods, OpenDR can best deform an input sphere to match the silhouette of a target teapot. However, none of these methods can handle topology changes (see the handle) and struggle with large deformation (see the spout). In comparison, DSS recovers these geometry structures with high fidelity and at the same time produces more elaborate surface details (see the pattern on the body of the teapot).

Finally, we compare with a naive point DR based on [Insafutdinov and Dosovitskiy 2018; Roveri et al. 2018a,b], where the pixel intensities are represented by the sum of smoothed depth values. As shown in Fig. 12, such a naive implementation of point-based DR cannot handle large-scale shape deformation nor fine-scale denoising, because position gradient is confined locally restricting long-range movement and normal information is not utilized to fine-grained geometry update.

5.2 Application: shape editing via image filter

As demonstrated in Paparazzi, one important application of DR is shape editing using existing image filters. It allows many kinds of geometric filtering and style transfer, which would have been challenging to define purely in the geometry domain. This benefit also applies to DSS.

We experimented with two types of image filters, L0 smoothing [Xu et al. 2011] and superpixel segmentation [Achanta et al. 2012]. These filters are applied to the original rendered images to create references. Like Paparazzi, we keep the silhouette of the

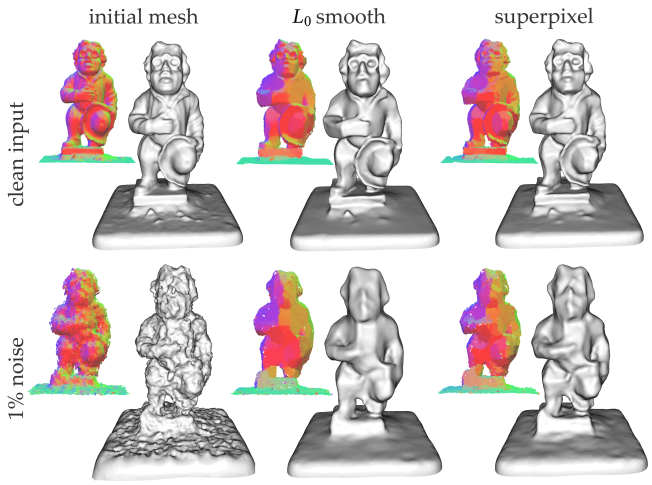


Fig. 13. Examples of DSS-based geometry filtering. We apply image filters on the DSS rendered multi-view images and propagate the changes of pixel values to point positions and normals. From left to right are the Poisson reconstruction of input points, points filtered by L_0-smoothing, and superpixel segmentation. In the first row, a clean point cloud is used as input, while in the second row, we add 1% white Gaussian noise. In both cases, DSS can update the geometry accordingly to match the changes in the image domain.

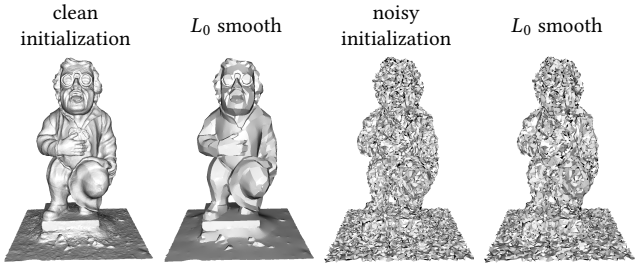


Fig. 14. Paparazzi [Liu et al. 2018] successfully applies a L_0 image filter to a clean mesh (Left) but fails on an input containing 0.5 % noise (Right).

shape and change the local surface geometry by updating point normals, then the projection and repulsion regularization are applied to correct the point positions.

As shown in Fig. 13, DSS successfully transfers image-level changes to geometry. Even under 1% noise, DSS continues to produce reasonable results. In contrast, mesh-based DRs are sensitive to input noise, because it leads to small piecewise structures and flipped faces in image space (see Fig. 14), which are troublesome for the computation of gradients. In comparison, points are free of any structural constraints; thus, DSS can update normals and positions independently, which makes it robust under noise.

5.3 Application: point cloud denoising

One of the benefits of the shape-from-rendering framework is the possibility to leverage powerful neural networks and vast 2D data. We demonstrate this advantage in a point cloud denoising task, which is known to be an ill-posed problem where handcrafted priors struggle with recovering all levels of smooth and sharp features.

First, we train an image denoising network based on the Pix2Pix [Isola et al. 2017] framework, which utilizes the generative adversarial network [Goodfellow et al. 2014] to add plausible details for improved visual quality (we refer readers to Appendix for further details on the training data preparation as well as the adapted network architecture). During test time, we render images of the noisy point cloud from different views and use the trained Pix2Pix network to reconstruct geometric structure from the noisy images. Finally, we update the point cloud using DSS with the denoised images as reference.

To maximize the amount of hallucinated details, we train two models for 1.0% and 0.3% noise respectively. Fig. 15 shows some examples of the input and output of the network. Hallucinated delicate structures can be observed clearly in both noise levels. Furthermore, even though our Pix2Pix model is not trained with view-consistency constraints, the hallucinated details remain mostly consistent across views. In case small inconsistencies appear in regions where a large amount of high-frequency details are created, DSS is still able to transfer plausible details from the 2D to the 3D domain without visible artefacts, as shown in Fig. 18, thanks to simultaneous multi-view optimization.

Evaluation of DSS denoising. We perform quantitative and qualitative comparison with state-of-the-art optimization-based methods WLOP [Huang et al. 2009], EAR [Huang et al. 2013], RIMLS [Öztireli et al. 2009] and GPF [Lu et al. 2018], as well as a learning-based method, PointCleanNet [Rakotosaona et al. 2019], using the code provided by the authors. For quantitative comparison, we compute Chamfer distance (CD) and Hausdorff distance (HD) between the reconstructed and ground truth surface.

First, we compare the denoising performance on a relatively noisy (1% noise) and sparse (20K points) input data, as shown in Fig. 17. Optimization-based methods can reconstruct a smooth surface but also smear the low-level details. The learning-based PointCleanNet can preserve some detailed structure, like the fingers of armadillo, but cannot remove all high-frequency noise. We test DSS with two image filters, i.e., the L_0 smoothing and the Pix2Pix model trained on data with 20K points and 1% noise. L_0-DSS has a similar performance with the optimization-based method. Pix2Pix-DSS outperforms the other compared methods quantitatively and qualitatively.

Second, we evaluate on a relatively smooth (0.3% noise) and dense (100K points) input data, as shown in Fig. 18. Optimization-based methods and L_0-DSS produce high-accuracy reconstruction. PointCleanNet's result deteriorates significantly, due to generalizability issues which is common for direct learning-based methods. In contrast, the proposed image-to-geometry denoising method is inherently less sensitive to the characteristic of points sampling. As a result, even though our Pix2Pix model is trained with 20K points, Pix2Pix-DSS reconstructs a clean surface, and at the same time shows abundant hallucinated details.

Finally, we evaluate Pix2Pix-DSS using real scanned data. We acquire a 3D scan of a dragon model by ourselves using a hand-held scanner and resample 50K points as input. We compare the point cloud cleaning performance of EAR, RIMLS, PointCleanNet and Ours as shown in Fig. 19. EAR outputs clean and smooth surfaces but

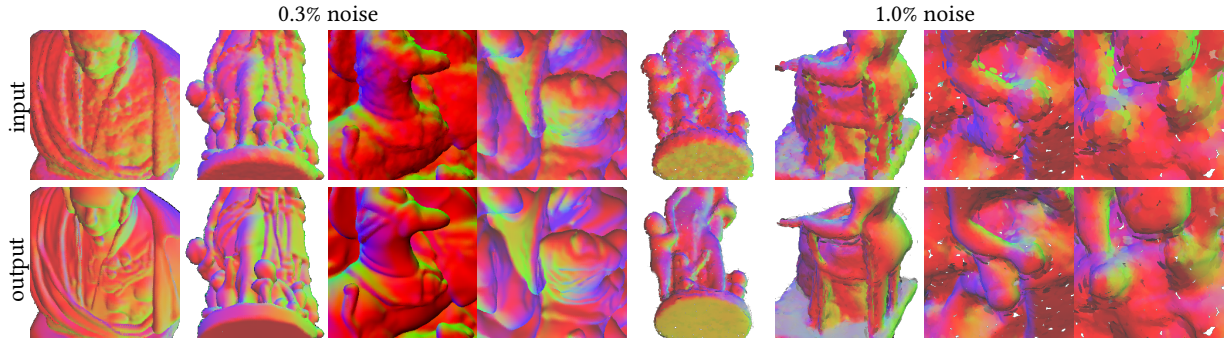


Fig. 15. Examples of the input and output of the Pix2Pix denoising network. We train two models to target two different noise levels (0.3% and 1.0%). In both cases, the network is able to recover smoothly detailed geometry, while the 0.3% noise variant generates more fine-grained details.

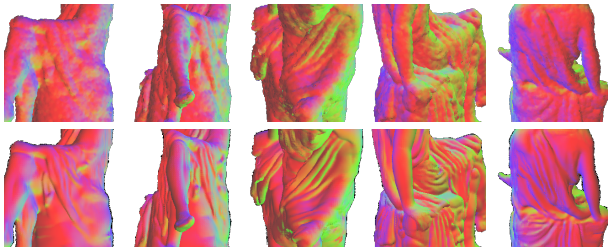


Fig. 16. Examples of multi-view Pix2Pix denoising on the same 3D model. As our Pix2Pix model processes each view independently, inconsistencies across different views might occur in the generated high-frequency details. In spite of that, DSS recovers plausible structures in the 3D shape (see Fig. 18) thanks to our simultaneous multi-view optimization.

tends to produce underwhelming geometry details. RIMLS preserves sharp geometry features, but compared to our method, its result contains more low-frequency noise. The output of PointCleanNet is notably noisier than other methods, while its reconstructed model falls between EAR and RIMLS in terms of detail preservation and surface smoothness. In comparison, our method yields clean and smooth surfaces with rich geometry details.

5.4 Performance

Our forward and backward rasterization passes are implemented in CUDA. We benchmark the runtime using an NVIDIA 1080Ti GPU with CUDA 10.0 and summarize the runtime as well as memory demand for all of the applications mentioned above on one exemplary model in Table 2. As before, models are rendered with 256×256 resolution and 12 views are used per optimization step.

As a reference, for the teapot example, one optimization step in Paparazzi and Neural Mesh Renderer takes about 50ms and 160ms respectively, whereas it takes us 100ms (see the second row in Table 2). However, since Paparazzi does not jointly optimize multiple-views, it requires more iterations for convergence. In the L0-Smoothing example (see Fig. 14), it takes 30 minutes and 30000 optimization steps to obtain the final result, whereas DSS needs 160 steps and 11 minutes for a similar result (see the third row in Table 2).

6 CONCLUSION AND FUTURE WORKS

We showed how a high-quality splat based differentiable renderer could be developed in this paper. DSS inherits the flexibility of point-based representations, can propagate gradients to point positions and normals, and produces accurate geometries and topologies. These were possible due to the careful handling of gradients and regularization. We showcased a few applications of how such a renderer can be utilized for image-based geometry processing. In particular, combining DSS with contemporary deep neural network architectures yielded state-of-the-art results.

There are a plethora of neural networks that provide excellent results on images for various applications such as stylization, segmentation, super-resolution, or finding correspondences, just to name a few. Developing DSS is the first step of transferring these techniques from image to geometry domain. Another fundamental application of DSS is in inverse rendering, where we try to infer scene-level information such as geometry, motion, materials, and lighting from images or video. We believe DSS will be instrumental in inferring dynamic scene geometries in multi-modal capture setups.

ACKNOWLEDGMENTS

We would like to thank Federico Danieli for the insightful discussion, Phillipp Herholz for the timely feedack and Romann Weber for the video voice-over. This work was supported in part by gifts from Adobe, Facebook and Snap, Inc.

REFERENCES

Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. 2012. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence* 34, 11 (2012), 2274–2282.

Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T Silva. 2003. Computing and rendering point set surfaces. *IEEE Trans. Visualization & Computer Graphics* 9, 1 (2003), 3–15.

Dejan Azinović, Tzu-Mao Li, Anton Kaplanyan, and Matthias Nießner. 2019. Inverse Path Tracing for Joint Material and Lighting Estimation. *arXiv preprint arXiv:1903.07145* (2019).

Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. 2017. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, Vol. 36. 301–329.

Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. 2008. MeshLab: an Open-Source Mesh Processing Tool.

model	application	number of points	total opt. steps for position	total opt. steps for normal	avg. forward time (ms)	avg. backward time (ms)	total time (s)	GPU memory (MB)
Fig. 10	shape deformation	8003	200	120	19.3	79.9	336	1.7MB
Fig. 13	L0 surface filtering	20000	8	152	42.8	164.6	665	1.8MB
Fig. 18	denoising	100000	8	152	258.1	680.2	1951	2.3MB

Table 2. Runtime and GPU memory demand for exemplar models in different applications. The images are rendered with 256×256 resolution and 12 views are used per optimization step.

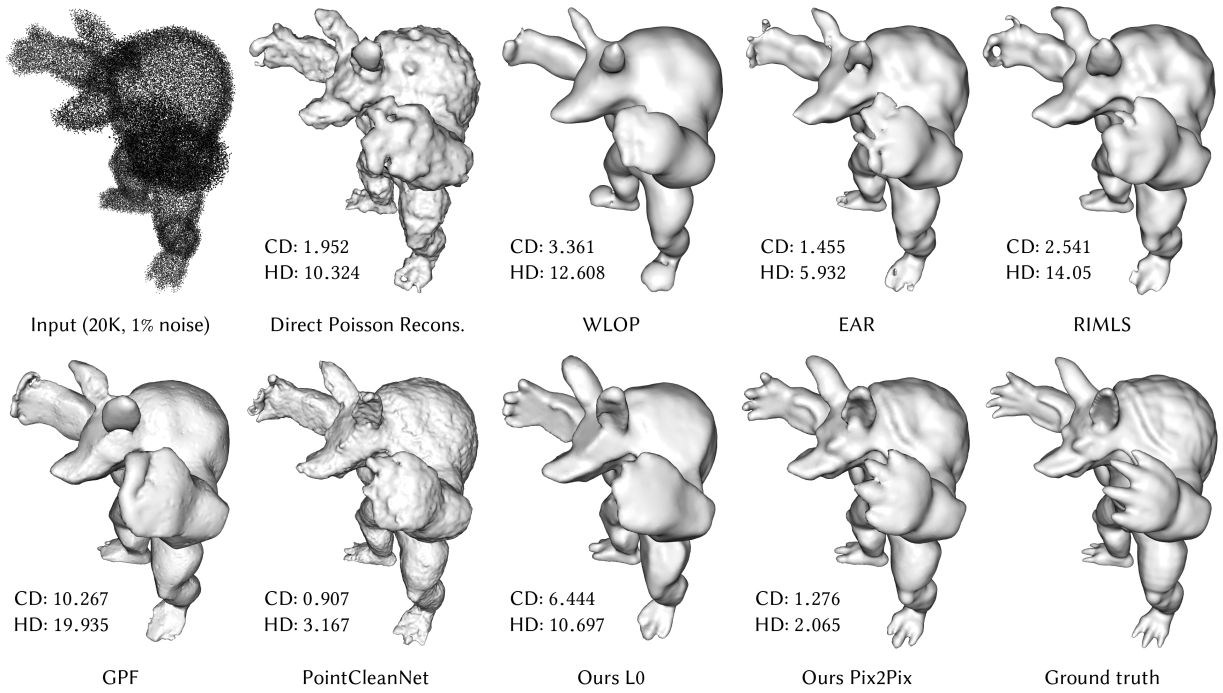


Fig. 17. Quantitative and qualitative comparison of point cloud denoising. The Chamfer distance (CD) and Hausdorff distance (HD) scaled by 10^{-4} and 10^{-3}. With respect to HD, our method outperforms its competitors, for CD only PointCleanNet can generate better, albeit noisy, results.

In *Eurographics Italian Chapter Conference*.

Massimiliano Corsini, Paolo Cignoni, and Roberto Scopigno. 2012. Efficient and flexible sampling with blue noise properties of triangular meshes. *IEEE Trans. Visualization & Computer Graphics* 18, 6 (2012), 914–924.

Haoqiang Fan, Hao Su, and Leonidas J Guibas. 2017. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. *Proc. IEEE Conf. on Computer Vision & Pattern Recognition* 2, 4, 6.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proc. Inter. Conf. on Artificial Intelligence and Statistics*. 249–256.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *In Advances in Neural Information Processing Systems (NIPS)*.

Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J Mitra. 2018. PCPNet learning local shape properties from raw point clouds. In *Computer Graphics Forum (Proc. of Eurographics)*, Vol. 37. 75–85.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*.

Paul S Heckbert. 1989. Fundamentals of texture mapping and image warping. (1989).

Pedro Hermosilla, Tobias Ritschel, and Timo Ropinski. 2019. Total Denoising: Unsupervised Learning of 3D Point Cloud Cleaning. *arXiv preprint arXiv:1904.07615* (2019).

P. Hermosilla, T. Ritschel, P-P Vazquez, A. Vinacua, and T. Ropinski. 2018. Monte Carlo Convolution for Learning on Non-Uniformly Sampled Point Clouds. *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia)* 37, 6 (2018).

Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1992. Surface reconstruction from unorganized points. *Proc. of SIGGRAPH* (1992), 71–78.

Tao Hu, Zhizhong Han, Abhinav Shrivastava, and Matthias Zwicker. 2019. Render4Completion: Synthesizing Multi-view Depth Maps for 3D Shape Completion. *arXiv preprint arXiv:1904.08366* (2019).

Hui Huang, Dan Li, Hao Zhang, Uri Ascher, and Daniel Cohen-Or. 2009. Consolidation of Unorganized Point Clouds for Surface Reconstruction. *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia)* 28, 5 (2009), 176:1–176:7.

Hui Huang, Shihao Wu, Minglun Gong, Daniel Cohen-Or, Uri Ascher, and Hao Richard Zhang. 2013. Edge-Aware Point Set Resampling. *ACM Trans. on Graphics* 32, 1 (2013), 9:1–9:12.

Eldar Insafutdinov and Alexey Dosovitskiy. 2018. Unsupervised learning of shape and pose with differentiable point clouds. In *In Advances in Neural Information Processing Systems (NIPS)*. 2802–2812.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. 2017. Image-To-Image Translation With Conditional Adversarial Networks. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *Proc. Int. Conf. on Learning Representations*.

Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. 2018. Neural 3d mesh renderer. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*. 3907–3916.

Fabian Langguth, Kalyan Sunkavalli, Sunil Hadap, and Michael Goesele. 2016. Shading-aware multi-view stereo. In *Proc. Euro. Conf. on Computer Vision*. Springer, 469–485.

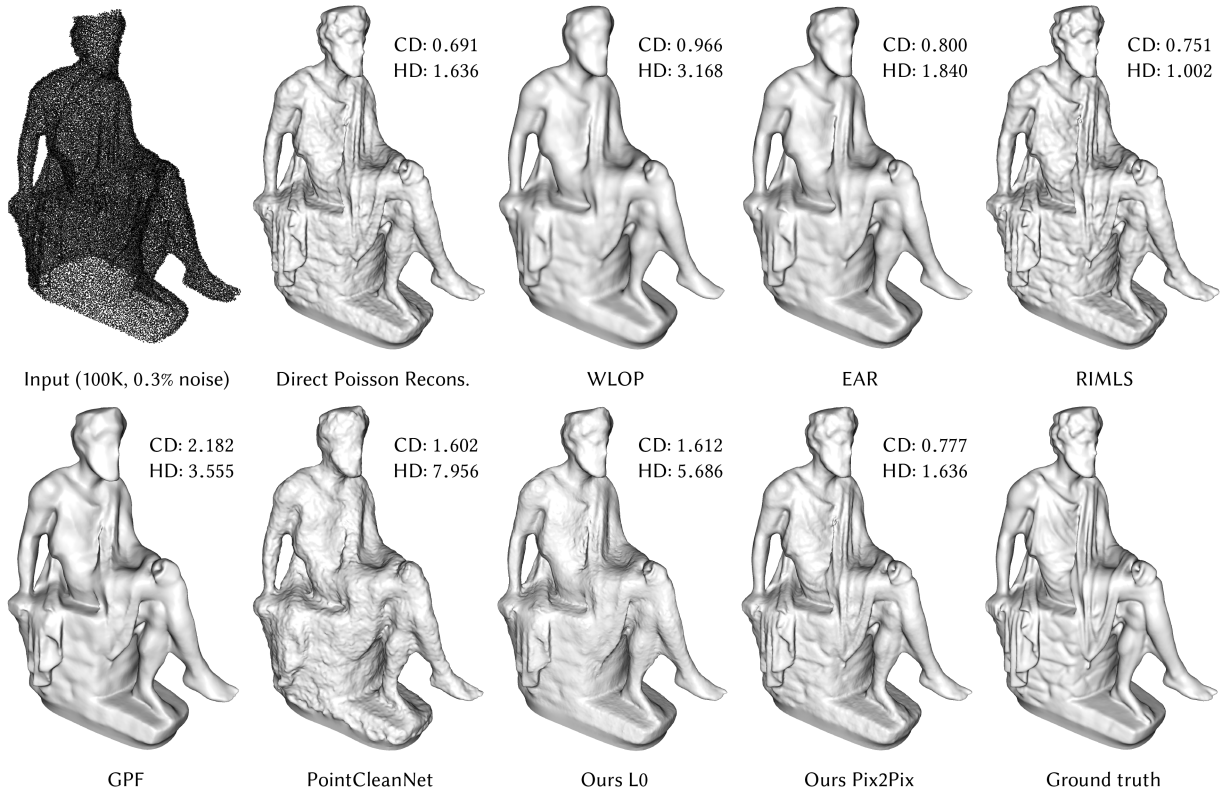


Fig. 18. Quantitative and qualitative comparison of point cloud denoising with 0.3% noise. We report CD and HD scaled by 10^{-4} and 10^{-3}. Despite some methods performing better with respect to quantitative evaluation, our result matches the ground truth closely in contrast to other methods.

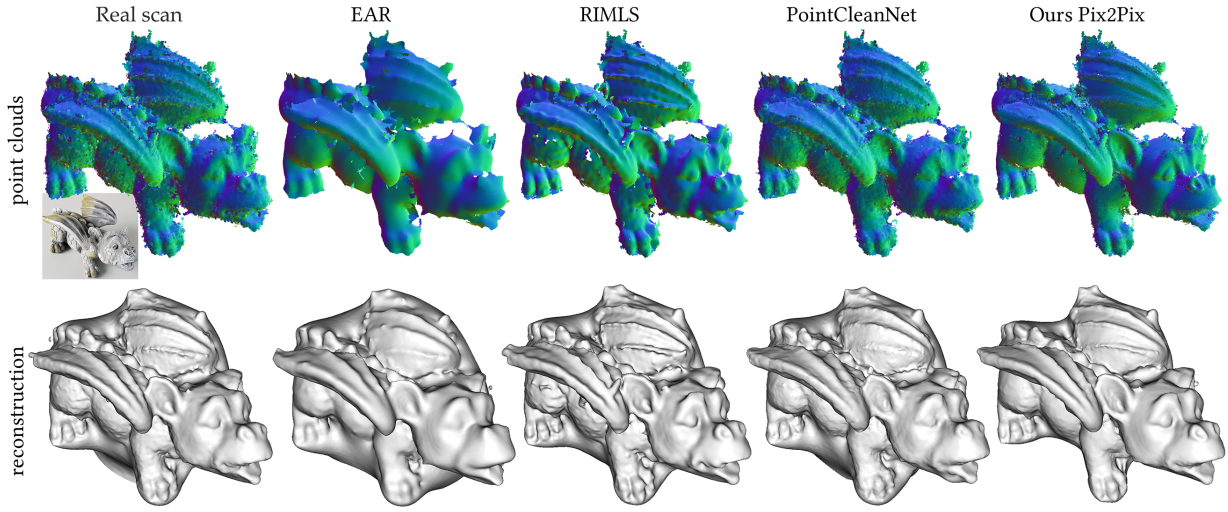


Fig. 19. Qualitative comparison of point cloud denoising on a dragon model acquired using a hand-held scanner (without intermediate mesh representation). Our Pix2Pix-DSS outperforms the compared methods.

Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018. Differentiable monte carlo ray tracing through edge sampling. In *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia)*. ACM, 222.

Chen-Hsuan Lin, Chen Kong, and Simon Lucey. 2018. Learning efficient point cloud generation for dense 3D object reconstruction. In *AAAI Conference on Artificial Intelligence*.

Yaron Lipman, Daniel Cohen-Or, David Levin, and Hillel Tal-Ezer. 2007. Parameterization-free projection for geometry reconstruction. *ACM Trans. on Graphics (Proc. of SIGGRAPH)* 26, 3 (2007), 22:1–22:6.

Guilin Liu, Duygu Ceylan, Ersin Yumer, Jimei Yang, and Jyh-Ming Lien. 2017. Material editing using a physically based rendering network. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*. 2261–2269.

Hsueh-Ti Derek Liu, Michael Tao, and Alec Jacobson. 2018. Paparazzi: Surface Editing by way of Multi-View Image Processing. In *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia)*. ACM, 221.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. 2019. Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning. *arXiv preprint arXiv:1904.01786* (2019).

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

Xuequan Lu, Shihao Wu, Honghua Chen, Sai-Kit Yeung, Wenzhi Chen, and Matthias Zwicker. 2018. GPF: GMM-inspired feature-preserving point set filtering. *IEEE Trans. Visualization & Computer Graphics* 24, 8 (2018), 2315–2326.

Robert Maier, Kihwan Kim, Daniel Cremers, Jan Kautz, and Matthias Nießner. 2017. Intrinsic3d: High-quality 3D reconstruction by joint appearance and geometry optimization with spatially-varying lighting. In *Proc. Int. Conf. on Computer Vision*. 3114–3122.

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. 2017. Least squares generative adversarial networks. In *Proc. Int. Conf. on Computer Vision*. 2794–2802.

Thu H Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yongliang Yang. 2018. RenderNet: A deep convolutional network for differentiable rendering from 3D shapes. In *In Advances in Neural Information Processing Systems (NIPS)*. 7891–7901.

A Cengiz Öztireli, Gael Guennebaud, and Markus Gross. 2009. Feature preserving point set surfaces based on non-linear kernel regression. In *Computer Graphics Forum (Proc. of Eurographics)*, Vol. 28. 493–501.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

Felix Petersen, Amit H Bermano, Oliver Deussen, and Daniel Cohen-Or. 2019. Pix2Vex: Image-to-Geometry Reconstruction using a Smooth Differentiable Renderer. *arXiv preprint arXiv:1903.11149* (2019).

Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. 2000. Surfels: Surface elements as rendering primitives. In *Proc. Conf. on Computer Graphics and Interactive techniques*. 335–342.

Jhony K Pontes, Chen Kong, Sridha Sridharan, Simon Lucey, Anders Eriksson, and Clinton Fookes. 2017. Image2Mesh: A Learning Framework for Single Image 3D Reconstruction. *arXiv preprint arXiv:1711.10669* (2017).

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*.

Sai Rajeswar, Fahim Mannan, Florian Golemo, David Vazquez, Derek Nowrouzezahrai, and Aaron Courville. 2018. Pix2Scene: Learning Implicit 3D Representations from Images. (2018).

Marie-Julie Rakotosaona, Vittorio La Barbera, Paul Guerrero, Niloy J Mitra, and Maks Ovsjanikov. 2019. POINTCLEANNET: Learning to Denoise and Remove Outliers from Dense Point Clouds. *arXiv preprint arXiv:1901.01060* (2019).

Elad Richardson, Matan Sela, Roy Or-El, and Ron Kimmel. 2017. Learning detailed face reconstruction from a single image. In *IEEE Trans. Pattern Analysis & Machine Intelligence*. 1259–1268.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *Inter. Conf. on Medical image computing and computer-assisted intervention*. Springer, 234–241.

Riccardo Roveri, A Cengiz Öztireli, Ioana Pandele, and Markus Gross. 2018a. Pointpronets: Consolidation of point clouds with convolutional neural networks. In *Computer Graphics Forum (Proc. of Eurographics)*, Vol. 37. 87–99.

Riccardo Roveri, Lukas Rahmann, Cengiz Oztireli, and Markus Gross. 2018b. A network architecture for point cloud classification via automatic depth images generation. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*. 4176–4184.

Soumyadip Sengupta, Angjoo Kanazawa, Carlos D Castillo, and David W Jacobs. 2018. SfSNet: Learning Shape, Reflectance and Illuminance of Facesin the Wild'. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6296–6305.

Jian Shi, Yue Dong, Hao Su, and Stella X. Yu. 2017. Learning Non-Lambertian Object Intrinsics Across ShapeNet Categories. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*.

Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. 2018. DeepVoxels: Learning Persistent 3D Feature Embeddings. *arXiv preprint arXiv:1812.01024* (2018).

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *Proc. IEEE Int. Conf. on Machine Learning*. 1139–1147.

Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. 2017. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*. 2626–2634.

Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with kernel prediction and asymmetric loss functions. *ACM Trans. on Graphics* 37, 4 (2018), 124.

Li Xu, Cewu Lu, Yi Xu, and Jiaya Jia. 2011. Image smoothing via L 0 gradient minimization. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 174.

Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. 2016. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *In Advances in Neural Information Processing Systems (NIPS)*. 1696–1704.

Shunyu Yao, Tzu Ming Hsu, Jun-Yan Zhu, Jiajun Wu, Antonio Torralba, Bill Freeman, and Josh Tenenbaum. 2018. 3D-aware scene manipulation via inverse graphics. In *In Advances in Neural Information Processing Systems (NIPS)*. 1887–1898.

Wang Yifan, Shihao Wu, Hui Huang, Daniel Cohen-Or, and Olga Sorkine-Hornung. 2018. Patch-based Progressive 3D Point Set Upsampling. *arXiv preprint arXiv:1811.11286* (2018).

Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. 2018. EC-Net: an Edge-aware Point set Consolidation Network. *Proc. Euro. Conf. on Computer Vision* (2018).

Rui Zhu, Hamed Kiani Galoogahi, Chaoyang Wang, and Simon Lucey. 2017. Rethinking reprojection: Closing the loop for pose-aware shape reconstruction from a single image. In *Proc. Int. Conf. on Computer Vision*. 57–65.

Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. 2002. Pointshop 3D: An interactive system for point-based surface editing. In *ACM Trans. on Graphics*, Vol. 21. ACM, 322–329.

Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. 2001. Surface splatting. In *Proc. Conf. on Computer Graphics and Interactive techniques*. ACM, 371–378.

Matthias Zwicker, Jussi Räsänen, Mario Botsch, Carsten Dachsbacher, and Mark Pauly. 2004. Perspective accurate splatting. In *Proc. of Graphics interface*. Canadian Human-Computer Communications Society, 247–254.

A PARAMETER DISCUSSION

Here, we describe the effects of all hyper-parameters of our method.

Forward rendering. The required hyper-parameters consist of C cutoff threshold, $\mathcal{T}$ merge threshold and σ_k (standard deviation). For all these parameters, we closely follow the default settings in the original EWA paper. [Zwicker et al. 2001]. For close camera views, the default C value is increased so that the splats are large enough to create hole-free renderings.

Backward rendering. The cache size K used for logging points which are projected to each pixel is the only hyper-parameter. The larger K is, the more accurate $\mathbf{W}_{\mathbf{x},k}$ becomes, as more occluded points can be considered for the re-evaluation of (6). We find $K = 5$ is sufficiently large for our experiments.

Regularization. Bandwidth $\mathcal{D}$ and Θ for computing weights in (12) and (13) are set as suggested in previous works [Huang et al. 2009; Öztireli et al. 2009]. Specifically, $\mathcal{D} = 4\sqrt{D/N}$, where D is the diagonal length of the bounding box of the initial shape and N is the number of points; Θ is set to $\pi/3$ to encourage a smooth surface under the presence of outliers. For large-scale deformation, where the intermediate results can have more outliers, we set $\mathcal{D}$ of the projection term to a higher value, e.g. $0.1\sqrt{D}$, which helps to pull the outliers to the nearest surface.

Optimization. The learning rate has a substantial impact on convergence. In our experiments, we set the learning rate for position and normal to 5 and 5000. These values generally work well for all applications. Higher learning rates cause the points to converge faster but increases the risk of causing the points to gather in clusters. A more sophisticated optimization algorithm can be applied for a more robust optimization process, but it is out of the scope of this paper. A sufficient number of views per optimization step is key to a good result in the ill-posed 2D-to-3D formulation. Twelve camera views are used in all our experiments, while with 8 or fewer

views results start to degenerate. The number of steps for points and normals update, $T_{\mathbf{p}}$ and $T_{\mathbf{n}}$, differ for each application. In general, for large topology changes, we set $T_{\mathbf{p}} > T_{\mathbf{n}}$, where typically $T_{\mathbf{p}} = 25$ and $T_{\mathbf{n}} = 15$, while for local geometry processing $T_{\mathbf{n}} > T_{\mathbf{p}}$ with $T_{\mathbf{n}} = 19$ and $T_{\mathbf{n}} = 1$. Finally, we find the loss weights for image loss $\mathcal{L}_{\mathbb{I}}$, projection regularization $\mathcal{L}_p$ and repulsion regularization $\mathcal{L}_r$, by ensuring the magnitude of per point gradient from $\mathcal{L}_p$ and $\mathcal{L}_r$ is around 1% of that from $\mathcal{L}_{\mathbb{I}}$. If the repulsion weight is too large, e.g. $\gamma_r > 0.1$, points can be repelled far off the surface, while if the projection weight is too large, e.g. $\gamma_p > 0.1$, points will be forced to stay on a local surface, making it difficult for topology changes.

B DENOISING PIX2PIX

Our model is based on Pix2Pix [Isola et al. 2017] that consist of a generator and a discriminator. For the generator, we experimented with U-Net [Ronneberger et al. 2015] and ResNet [He et al. 2016], and find ResNet performs slightly better in our task, which we use for all experiments. That is, the generator has a 2-stride convolution and a 2-stride up-convolution for both the encoder and decoder networks and 9 residual blocks in-between. The discriminator follows the architecture as: C64-C128-C256-C512-C1, where LSGAN [Mao et al. 2017] is used. To deal with checkerboard artifacts, we use pixel-wise normalization in the generator and add a convolutional layer after each deconvolutional layer in the discriminator [Karras et al. 2018]. Furthermore, we remove the `tanh` activation in the final layer in order to obtain unbounded pixel values We use the default parameters of the Pix2Pix pytorch implementation provided by the authors, and ADAM optimizer ($lr = 0.0002, \beta_1 = 0.5, \beta_2 = 0.999$) . Xavier [Glorot and Bengio 2010] is used for weights initialization. We train our models for about two days on an NVIDIA 1080Ti GPU.

To synthesize training data for the Pix2Pix denoising network, we use the training set of the Sketchfab dataset [Yifan et al. 2018], which consist of 91 high-resolution 3D models. We use Poisson-disk sampling [Corsini et al. 2012] implemented in Meshlab [Cignoni et al. 2008] to sample 20K points per model as reference points, and create noisy input points by adding white Gaussian noise, then we compute the PCA normal [Hoppe et al. 1992] for both the reference and input points. We generate training data by rendering a total of 149240 pairs of images from the noisy and clean models using DSS, from a variety of viewpoints and distances. We use point light and diffuse shading. While using sophisticated lighting, non-uniform albedo and specular shading can provide useful cues for estimating global information such as lighting and camera positions, we find the glossy effects pose unnecessary difficulties for the network to infer local geometric structure.