Pointersect: Neural Rendering with Cloud-Ray Intersection
Supplementary Material

Jen-Hao Rick Chang[1], Wei-Yu Chen[1,2]*, Anurag Ranjan[1], Kwang Moo Yi[1,3]*, Oncel Tuzel[1]
[1]Apple, [2]Carnegie Mellon University, [3]University of British Columbia
https://machinelearning.apple.com/research/pointersect

In the supplementary material, we provide details about the following topics:

- *overview of related work* in Appendix A;
- *accelerated structure* in Appendix B;
- *model architecture* in Appendix C;
- *complexity and runtime analyses* in Appendix D;
- *training procedure* in Appendix E;
- *inverse rendering* in Appendix F;
- *Chamfer distance* in Appendix G;
- *additional results* to evaluate and gain insights on pointersect, including:
  - novel-view rendering video and additional scenes in Figure 13 and in the offline web page,
  - ablation study on the number of input views and the choice of k in Appendix H.1,
  - ablation study on the input resolution in Appendix H.2,
  - and the effect of ground-truth vertex normal to Poisson reconstruction in Appendix H.3;
- *Effect of noise in depth map* in Appendix I;
- and finally, the *entire training dataset* containing 48 meshes and their credits in Appendix J.

A. Additional related work

In this paper, we focus on comparing with point-cloud rendering methods that do not require per-scene optimization. In this section, we briefly discuss and include additional related work. We also provide an overview in Tab. 4 for interested readers.

Many recent works develop novel view synthesis techniques given only the RGB images and optionally their camera information. Neural Radiance Field (NeRF) [22] achieve a great success by representing the volume density and the radiance field with a neural network, and it is able to render high-quality photo-realistic images from novel viewpoints. Many follow-up works further improve the quality [3, 42], the training efficiency [33, 46], the rendering speed [11, 23], and the generalization capability [43, 47]. Additional capabilities have also been introduced into NeRF, including estimating shape and reflectance [5, 39, 48]. Several methods also aim to reduce or completely avoid the per-scene optimization of NeRF by utlizing image features [6, 43, 47]. These methods often rely on additional information, including class labels,

Other rendering primitives, including spheres [17], occupancy field [25], Signed Distance Function (SDF) [15, 45], light field [38, 40], or a plane sweep volume [6, 7, 21] have also been developed.

Hybrid methods have also been developed. These methods first utilize Structure from Motion (SfM) or multi-view geometry to estimate scene geometry form the input RGB images. The estimated geometry is then used to provide additional supervision for NeRF [9, 36] or to anchor feature aggregation [34, 35].

As mentioned earlier, this paper focuses on point-cloud rendering *without* per-scene optimization. The methods discussed above operate in a different setting or require per-scene training. Therefore, while we are inspired by many of these methods, our method is not directly comparable.

B. Finding points along a ray

Finding the intersection point of a ray $\mathbf{r} = (r_o, \vec{r_d})$ with a point cloud $\mathcal{P}$ requires only points near the ray. Thus, we pass only the k nearest points within a cylinder of radius δ in terms of their perpendicular distances to the ray. A naive implementation would examine all points in $\mathcal{P}$, sort them according to their distances, and keep the k nearest ones, taking $\mathcal{O}(n \log n)$ operations per ray, where n is the total number of points in $\mathcal{P}$. Since we only need points with the cylinder, a commonly used strategy to reduce the time complexity is to build an accelerated structure like octree to reduce the number of candidate points to examine [29]. Building an octree takes $\mathcal{O}(n \log n)$ operations, and searching for nearby points takes $\mathcal{O}(\log n)$ operations per ray. For a static scene, the octree can be built once and keep

*Work done at Apple. Corresponding author: jenhao_chang@apple.com

Table 4. **Rendering methods for 2D images and point clouds.** The table provides a summary of the scene presentation (geometry primitives) and the capabilities of various methods. ✓*: require vertex normal as inputs. ✓$^\flat$: provides good results, and per-scene optimization further improves the quality. ✓: depth and normal can be estimated from the density function.

Category	Method	Primitives	Render color	Estimate depth	Estimate normal	No per-scene optimization
Rendering from 2D images	NeRF [22]	volume density (MLP)	✓	✓	✓	✗
	NeRD [5]	volume density (MLP)	✓	✓	✓	✗
	NeRV [39]	volume density (MLP)	✓	✓	✓	✗
	NeRFactor [48]	volume density (MLP)	✓	✓	✓	✗
	KiloNeRF [33]	volume density (grid MLP)	✓	✓	✓	✗
	InstantNGP [23]	volume density (multi-resolution MLP)	✓	✓	✓	✗
	Plenoctrees [46]	volume density (octree)	✓	✓	✓	✗
	Plenoxels [11]	volume density (voxel)	✓	✓	✓	✗
	PixelNeRF [47]	volume density (MLP + 2D features)	✓	✓	✓	✓
	IBRNet [43]	volume density (MLP + 2D features)	✓	✓	✓	✓$^\flat$
	Pulsar [17]	spheres	✓	✓	✓	✗
	Differentiable Volumetric Rendering [25]	occupancy field	✓	✓	✓	✗
	IDR [45]	SDF	✓	✓	✓	✗
	Neural Lumigraph Rendering [15]	SDF	✓	✓	✓	✗
	Light field Networks [38]	light field	✓	✗	✗	✗
	Light Field Neural Rendering [40]	light field	✓	✗	✗	✗
	Extreme View Synthesis [7]	plane sweep volume	✓	✓	✗	✓
	MVSNeRF [6]	plane sweep volume	✓	✓	✓	✓$^\flat$
	Local Light Field Fusion [21]	plane sweep volume + light field	✓	✓	✗	✓
Rendering from 2D images + SfM Depth	Depth-supervised NeRF [9]	volume density (MLP)	✓	✓	✓	✗
	Dense Depth Priors [36]	volume density (MLP)	✓	✓	✓	✗
	Free View Synthesis [34]	mesh	✓	✓	✓	✓
	Stable View Synthesis [35]	mesh	✓	✓	✓	✓
Point cloud to other representations	Poisson reconstruction [14]	indicator func.	✓	✓	✓*	✗
	Shape as point [27]	indicator func.	✗	✓	✓*	✗
	Neural pull [18]	SDF	✗	✓	✓	✗
	Point2Mesh [12]	mesh	✗	✓	✓	✗
	Neural point [10]	neural point	✗	✓	✓	✓
Point cloud rasterization	Visibility Splatting [28]	surfel	✓	✓	✓*	✓
	NPBG [2]	points	✓	✗	✗	✗
	ADOP [37]	points	✓	✗	✗	✗
	Multi-plane [8]	points	✓	✗	✗	✗
	NPBG++ [31]	points	✓	✗	✗	✓
Point cloud ray-casting	Iterative ray-surface intersection [1]	points	✓	✓	✓	✗
	Point-Nerf [44]	points	✓	✓	✓	✓$^\flat$
	NPLF [26]	points	✓	✗	✗	✗
	Ours (pointersect)	points	✓	✓	✓	✓

reusing the tree to find neighbor points.

While octree greatly improves the speed for static $\mathcal{P}$, for dynamic scenes or for inverse rendering, where many points in $\mathcal{P}$ can change at every iteration, rebuilding the tree at every iteration becomes an overhead. We build an accelerated structure that can be built in parallel on GPU to improve the speed of inverse rendering. The structure is based on a voxel grid. Intuitively, as can be seen from Fig. 8, we divide the space into voxels, and we record the points contained in each voxel with a table. Thus, building the voxel grid structure takes $\mathcal{O}(n)$ operations, where n is the total number of points. When a ray is given, we trace the ray through the voxel grid. The grid-ray intersection can be computed in $\mathcal{O}(g)$ time for a given ray, where g is the number of grid cells per dimension, and the computation for each ray can be computed in parallel using multiple GPU threads.

Overall, the time complexity of finding k nearest points within a surrounding cylinder of radius δ is $\mathcal{O}\left(n + mg + mq\log(q)\right)$, where q is the number of points lying within the cylinder, and the last term corresponds to the cost of sorting the distances from the points to the ray. In the case, when the points are uniformly distributed, in expectation $q = n/g^3$ Note that our implementation is not optimized; for example, the optimal time complexity of retrieving an unordered k nearest points from the q points is $\mathcal{O}(q)$. This structure can be slower than using an octree for static scenes, which takes $\mathbf{n}\log(n) + m\log(n) + mq$, but its faster construction makes it more suitable for dynamic point clouds. We also want to point out that while we implement the accelerated structure in CUDA, the implementation is not optimized, and the speed can be further improved, as we will see in Appendix D.

C. Architecture details

We use a transformer to build the pointersect model. The model architecture details, including the number of layers and the layer composition, are illustrated in Fig. 9b.

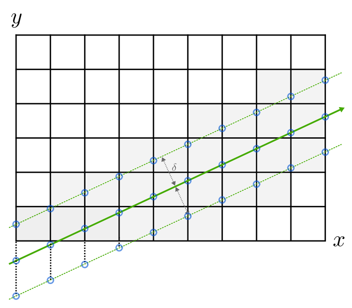


Figure 8. The voxel grid structure utilizes the easy computation of grid-ray intersection. Note that we need to trace the ray along the slowest moving axis (in this case the $x = c_i$ planes).

The model architecture is composed of a Multi-layer Perceptron (MLP) and a transformer. The MLP is used to convert the input features (*e.g.*, xyz, rgb) into the dimensionality used by the transformer (which is 64). We do not use positional encoding for xyz like NeRF [22]—we found that adding positional encoding reduces the estimation accuracy.

We use the standard transformer block from Vaswani et al. [41]. We remove the layer normalization layer, and we use a dropout probability of 0.1. The transformer is composed of 4 layers of transformer blocks and has a dimension of 64. We additional learn a token and insert it at the input of the transformer, in order to estimate the ray traveling distance, the surface normal, and the probability of hitting a surface. We use the SiLU nonlinearity [13, 32].

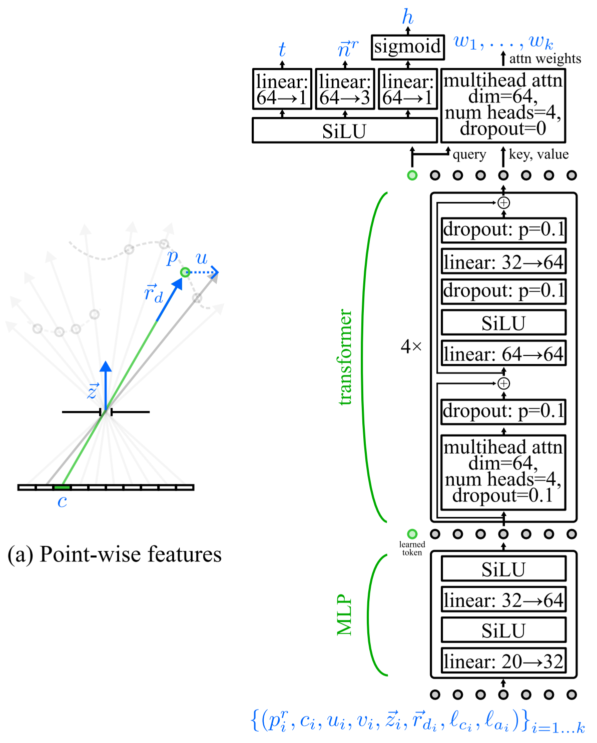
The output of the transformer contain $k + 1$ tokens—1 corresponding to the special token and k for the neighboring points. We use a linear layer to convert the special token to the corresponding estimates. To compute the material blending weights, $\mathbf{w} = \left[w_1, \ldots, w_k \,|\, w_i \in [0,1], \sum_{i=1}^{k} w_i = 1\right]$, we use a multihead attention layer and take its output attention weights as $\mathbf{w}$. In other words, the tokens are projected by a linear layers as queries, keys, and values, and the softmax attention is used to compute the similarities between the tokens and the special token.

D. Complexity analysis

In this section, we analyze the computation complexity of our implementation. Pointersect is composed of three main operations:

- finding points along the query ray (Appendix B);
- transform to the canonical coordinate (Section 3);
- and run the transformer model (Appendix C).

Finding neighboring points, as discussed in Appendix B, has a time complexity of $\mathcal{O}(n + mg + mq\log(q))$, where n is the number of points, m is the number of query rays, g is the number of cells in each dimension of the grid, and



(a) Point-wise features

(b) Model architecture details

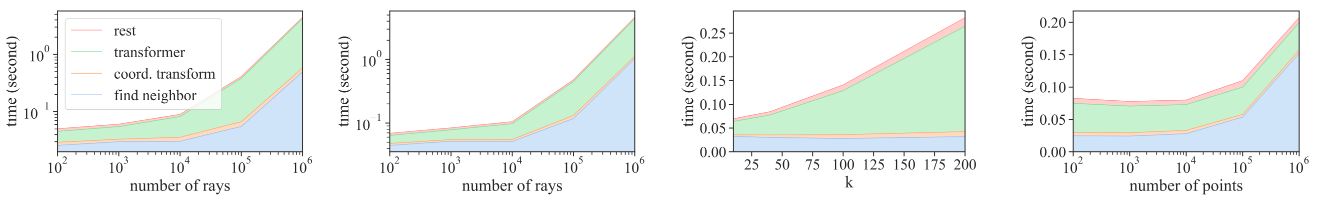
Figure 9. (a) Point-wise features extracted from camera rays and (b) model architecture details.

q is the total number of points lying within a cylinder of radius δ centered on the query ray. In the case, when the points are uniformly distributed, in expectation $q = n/g^3$ The transformation to the canonical coordinate is $\mathcal{O}(mk)$, where k is the number of the nearest neighbor points. The transformer has a time complexity of $\mathcal{O}(Lkd(k+d))$, where L and d are the number of layers and the dimension of the transformer, respectively.

Fig. 10 shows the runtime measurements of various settings. In the experiment, we simulate an adversarial scenario where the points are uniformly sampled within a square cube between $[-1, 1]^3$ (instead of on surfaces as typical scenarios). As can be seen from the results, the main bottlenecks are finding neighboring points of a ray and the transformer. While we implement the neighbor-point searching with a custom CUDA kernel, there is still a significant room for improvement. Nevertheless, with a point cloud containing $n = 10,000$ points and $k = 40$, our non-optimized implementation is capable of rendering 100×100 images in 10 fps, 200×200 images in 5 fps, and 500×400 in 1 fps.

E. Model training details

We train the pointersect model with the 48 training meshes in the sketchfab dataset [30]. We center and scale



(a) effect of number of rays. number of points = 10,000 $k = 40$

(b) effect of number of rays. number of points = 100,000 $k = 40$

(c) effect of k. number of points = 10,000 number of rays = 10,000

(d) effect of number of points. number of rays = 10,000 $k = 40$

Figure 10. Runtime measurements of the proposed method rendering (a, b) different numbers of rays, (c) different numbers of neighboring points k, and (d) different number of points. We break the total runtime into four main categories and illustrate them as cumulative plots. In the experiment, we uniformly sample points within a square cube between $[-1, 1]^3$, uniformly sample ray origin in the cube, and uniformly sample rays toward all directions. We repeat each case by three times and report the average. The experiment is conducted on a single A100 GPU and PyTorch 1.10.1 with the typical implementation of transformer.

the meshes such that the longest side of their bounding box is 2 units in length. For each training iteration, we randomly select one mesh and randomly construct 30 input cameras and 1 target camera, which capture RGBD images using the mesh-ray intersection method in Open3D [50]. We do not apply anti-aliasing filters on the RGBD images. This allows us to get the ground truth of the specific intersection point, instead of a blurred and average one across a local neighborhood. The ground-truth RGBD images are rendered without global illumination. This is intentional and allows the learned $\mathbf{w}$ to focus on material properties and not be affected by lighting conditions and cast shadows. Both input and target cameras have a field-of-view of 60 degrees.

We create the input point cloud using the input RGBD images. Specifically, for each pixel in an input RGBD image, we cast a ray from the pixel center towards the camera pinhole and use the depth map to determine the point location. For each point, we gather the following information (see Fig. 9a for illustration):

- *xyz*, or $p \in \mathbb{R}^3$, the location of the point;
- *rgb*, or $c \in \mathbb{R}^3$, color of the point in input RGBD image;
- $\vec{r}_d \in \mathbb{S}^2$, the cast camera ray direction (normalized);
- $u \in \mathbb{R}^3$, a vector from the point corresponding to the current pixel to the point corresponding to the next pixel in the x direction on the input RGBD image (see Fig. 9a);
- $v \in \mathbb{R}^3$, same as u but in the y direction;
- $z \in \mathbb{S}^2$, the optical axis direction of the input camera;
- $\ell_c \in \{0, 1\}$, a binary indicator, which is set to 1 when c contains valid information or 0 when we set c to $(0.5, 0.5, 0.5)$;
- and $\ell_a \in \{0, 1\}$, a binary indicator, which is set to 1 when $\vec{r}_d, u, v, v, z$ contain valid information or 0 when we set all of them to zeors.

All of the features are point-wise and can be easily extracted from the camera pose. To support point clouds that contain only xyz information and without these information, we randomly drop rgb and other features independently 50 % of the time (and set ℓ_c and ℓ_a accordingly). During inference, in all experiments shown in the paper, we do not use any of the features, except xyz and rgb.

To diversify the sampling rates of the point cloud, we randomly set the resolution (ranging from 30×30 to 300×300) and position (1 to 3 units from the origin) for each input camera. The target camera has a 50×50 resolution, *i.e.*, 2500 query rays per iteration. To increase the diversity of the query rays, we point the target camera towards a random point in a centered box of a width equal to 1 unit and position the camera at a random location (between 0.5 to 3 units to the origin). To help learning the blending weights of color, at every iteration we select a random image patch with a size from 20×20 to 200×200 in the ImageNet dataset as the texture map for the mesh. We also select a random $k \in [12, 200]$ at every iteration.

To optimize the loss function in Eq. (1), We use ADAM [16] with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and a learning rate schedule used by Vaswani et al. [41] with a warm-up period of 4,000 iterations. Within the warm-up iterations, the learning rate increases rapidly to $2e^{-6}$, and it gradually decreases afterwards. We train the model for 350,000 iterations, and it takes 10 days on 8 A100 GPUs.

F. Inverse rendering: details

As we have discussed, the pointersect model f allows gradient computation of color and surface normal with respect to the point cloud's xyz and rgb. In this section, we provide details of our inverse rendering experiment in Section 4.5. Our goal is to demonstrate the use of pointersect in an inverse rendering application, so we assume a simple scenario where we have ground-truth RGB images, camera poses, and foreground segmentation masks. Our purpose is to demonstrate the potential, not to compare with the state-of-the-art inverse rendering methods.

Given N input RGB images, $\mathcal{I} = \{I_1, \ldots, I_N | I_i \in \mathbb{R}^{h\times w\times 3}\}$, their corresponding foreground masks, $\mathcal{Y} = \{Y_1, \ldots, Y_N | Y_i \in [0,1]^{h\times w}\}$, camera extrinsic and intrinsic matrices, and a noisy point cloud, $\mathcal{P} = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, our goal is to optimize the position $\mathbf{p} = \{p_1, \ldots, p_n\}$ and color $\mathbf{c} = \{c_1, \ldots, c_n\}$ of the points such that when we render $\mathcal{P}$ with pointersect from the input camera views, the output images match the input ones.

Let $\mathbf{r}_k^j$ be a camera ray from the j-th input image. We use bilinear interpolation to calculate the corresponding color and the foreground mask values, $\hat{c}(\mathbf{r}_k^j)$ and $\hat{y}(\mathbf{r}_k^j)$, respectively. We optimize the following loss function while fixing the network parameter of the pointersect model (*i.e.*, simply use it as part of the rendering forward function):

$$\min_{\mathcal{P}} \sum_{j=1}^{N} \sum_{\mathbf{r}_k^j} \left\| \hat{c}(\mathbf{r}_k^j) - c(\mathbf{r}_k^j, \mathcal{P}) \right\|_2^2 \tag{1}$$

$$- \hat{y}(\mathbf{r}_k^j) \log h(\mathbf{r}_k^j, \mathcal{P}) \tag{2}$$

$$- (1 - \hat{y}(\mathbf{r}_k^j)) \log\Big(1 - h(\mathbf{r}_k^j, \mathcal{P})\Big) \tag{3}$$

$$+ \Big(n(\mathbf{r}_k^j, \mathcal{P}) \times n(\mathbf{r}_{k+1_u}^j, \mathcal{P})\Big)^2 \tag{4}$$

$$+ \Big(n(\mathbf{r}_k^j, \mathcal{P}) \times n(\mathbf{r}_{k+1_v}^j, \mathcal{P})\Big)^2, \tag{5}$$

where $c(\mathbf{r}_k^j, \mathcal{P})$, $n(\mathbf{r}_k^j, \mathcal{P})$, and $h(\mathbf{r}_k^j, \mathcal{P})$ are pointersect's estimates of color (using blending weights), surface normal, and hit, respectively (see Section 3 in the main paper). In the optimization program, we minimize the ℓ_2 loss of color (eq. (1)), the negative log-likelihood of foreground hit estimation (eq. (2) and eq. (3)), and the smoothness of the estimated normal map between neighboring pixels on the images (eq. (4) and eq. (5)). The loss terms related to normal smoothness is optional—we add it for demonstration. Note that we do not couple the input rgb colors with the ground-truth surface normal to provide additional supervision to the geometry—the color is computed simply by interpolating the rgb values of input images.

We create an example problem by capturing 100 RGBD images from the Stanford Bunny mesh [49] (whose size is scaled to have 2 units in length), adding Gaussian noise with standard deviation equal to 0.2 to the depth channel, and using the RGBD images to create the noisy point cloud and RGB images as our input images. We solve the optimization problem using stochastic gradient descent with a learning rate of 0.01 with 10000 iterations. Each iteration we select a point in $\mathcal{P}$, project it onto all input images, and cast rays from the local 10×10 patch of the projected pixel location. We cast roughly a total of 10000 rays every iteration. At the end of every iteration, we project the RGB values of the point cloud to $[0, 1]$. Every 150 iterations, we use the ground-truth foreground segmentation maps to perform silhouette carving on $\mathcal{P}$, which allows us to remove points that are apparently invalid. We insert new points by casting rays from random input pixels using pointersect's estimates of point position and color. This operation is fast, easy to implement, and significantly speeds up the optimization. The entire optimization (10000 iterations) takes 1 hour on one A100 GPU.

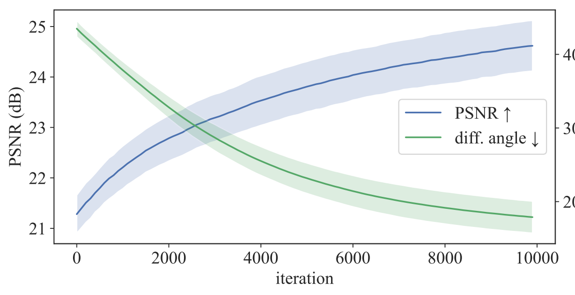


Figure 11. PSNR and the angle between the ground-truth and estimated color and surface normal, respectively, during the inverse rendering optimization. The shade represents the standard deviation of the values over the 100 input views.

Table 5. Optimize a noisy point cloud with pointersect.

	100 input views		144 novel views	
	before opt.	after opt.	before opt.	after opt.
PSNR (dB) ↑	10.1 ± 0.5	24.6 ± 0.5	13.9 ± 0.8	25.7 ± 1.2
normal (angle (°)) ↓	54.0 ± 0.9	17.8 ± 2.1	55.3 ± 0.4	18.3 ± 2.7
depth (rmse) ↓	0.46 ± 0.08	0.09 ± 0.06	0.33 ± 0.08	0.10 ± 0.07

Figure 12 shows the results, Fig. 11 shows the optimization progression, and Table 5 shows the statistics on 100 input views and 144 novel views. As can be seen, with the capability to back-propogate gradient through pointersect, we are able to optimize the point cloud to reduce the rgb, depth, and normal errors.

G. Deviation from ground-truth surface

We measure the error "along" the ray as we are interested in the accuracy from a ray intersection stand point, which is the main focus of the paper. To further provide insight into the accuracy of the surfaces that would be reconstructed, Table 6 shows the Chamfer distance using the same setup as Table 1.

Table 6. Chamfer distance ($\times 10^{-3}$). Note that Chamfer distance measures square distances, whereas the RMSE (used by the rest of the paper) measure distance.

dataset	Visibility splatting	Poisson recon.	Neural Points	Ours
ShapeNet	2.32 ± 9.55	0.51 ± 1.98	0.30 ± 0.43	$\mathbf{0.29 \pm 0.48}$
Sketchfab	13.33 ± 25.53	1.58 ± 4.30	0.98 ± 2.47	$\mathbf{0.87 \pm 2.19}$

H. Additional results

Figure 13 extends Figure 4 in the paper and shows two results from the Sketchfab test dataset. The supplementary

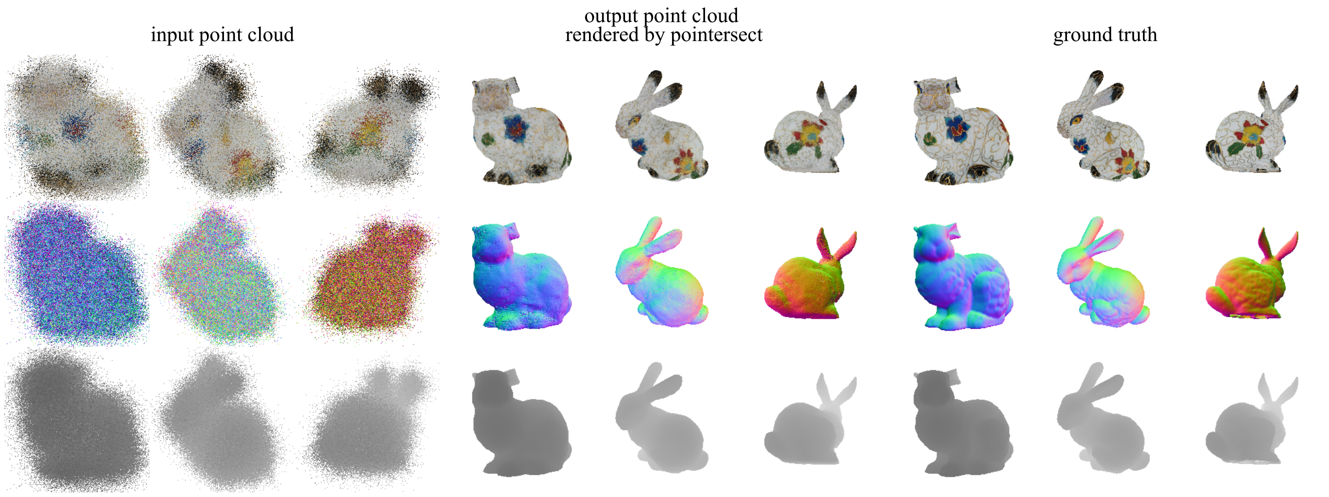


Figure 12. We back-propagate gradient through the pointersect model to optimize a noisy point cloud, comparing output images with the given clean RGB images and foreground binary mask.

offline website showcases the result videos of Section 4.2, Section 4.3, Section 4.5, and Section 4.6. In the following, we provide two ablation studies on the number of views, the choice of k, and the sampling rate (*i.e.*, density) of the point clouds.

H.1. Number of input views and the choice of k

In Section 4.2, we constructed the input point clouds by capturing 6 200×200 RGBD images, each from front, back, left, right, top, and bottom, of the object of interest. The input point cloud, constructed in this manner, may fail to contain occluded part of the object. Here, we repeat the experiment with an increased number of input RGBD images, 30 and 60. Specifically, we randomly sample 30 (or 60) cameras uniformly within a sphere shell with inner radius of 3 and outer radius of 4. All cameras have a field of view of 30 degrees, a resolution of 200×200, and point to the center of the object of interest. All settings are the same as those in Section 4.2—$k = 40$, $\delta = 0.1$, and we provide ground-truth vertex normal to Poisson reconstruction and visibility splatting. The depth and normal errors are computed only on non-hole pixels. Additionally, since increasing the number of input views increases the density of the point cloud, choosing the same number of nearest points, *i.e.*, using the same k, effectively reduces the size of the neighborhood the pointersect model can attend to. Thus, we also include a pointersect result where we increase k to 200, which retains the expected neighborhood size for 30 input views and halves it for 60 input views.

The results are shown in Tab. 7. As can be seen, pointersect, regardless of the choice of k, achieves the best results in color estimations. Increasing k (thus allows pointersect to attend to a larger neighborhood) also makes pointersect outperform all baselines in terms of the estimation accuracy of depth and surface normal.

H.2. Number of views *vs*. point density

In this experiment, we study the effect of point density when we capture the scene in a frontal-view position. We take the Stanford Bunny [49] and capture 1 RGBD image from the center frontal view in various resolutions. Since the field-of-view of remain the same, increasing the resolution increases the density of the points. We also capture 4 more RGBD images at the 4 corners of the axis-aligned bounding box, also from the frontal view. The output cameras are on a circle centered at the center viewpoint and looking towards the center of the bunny. All settings are the same as those in Section 4.2, *i.e.*, $k = 40$, $\delta = 0.1$, and we provide the ground-truth vertex normal to Poisson surface reconstruction and visibility splatting.

The results are shown in Table 8 and Figure 14. As can be seen, even with a single view point, pointersect is able to render the point cloud from novel view points. Increasing the point density improves the estimation accuracy of pointersect. When we use only a single view (and part of the bunny is occluded and missing in the input point cloud), pointersect renders faithfully the point cloud and thus produces with a missing ear. The occluded ear appears when we include all 5 input images.

H.3. Poisson without ground-truth vertex normal

In all experiments in the paper (except the ones on real Lidar point cloud where we do not have ground-truth vertex normal), we provide the ground-truth vertex normal to Poisson surface reconstruction. However, in practice, ground-truth vertex normal is difficult to get, and thus we often need to estimate the vertex normal from the point cloud before computing Poisson reconstruction.

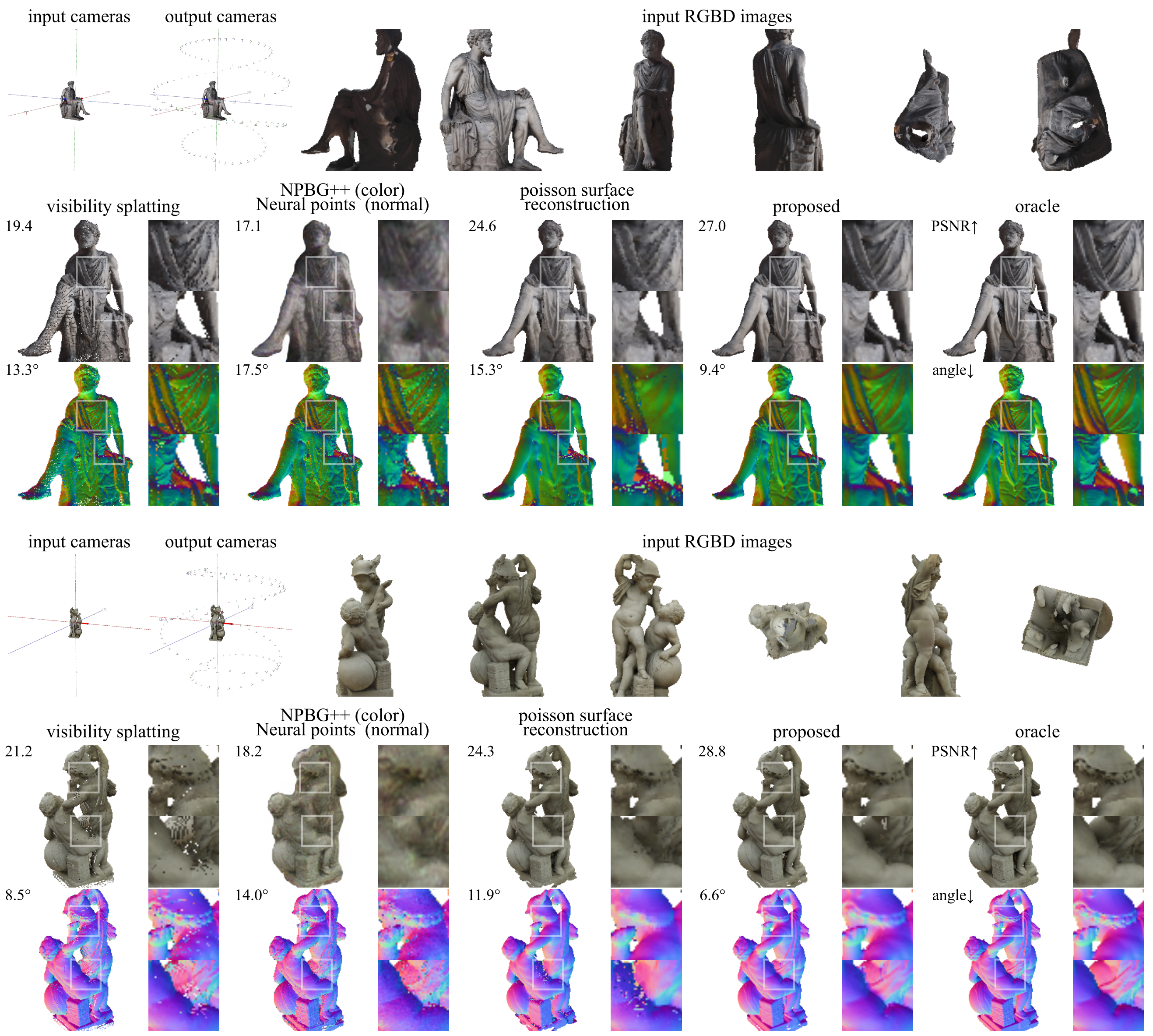


Figure 13. Additional example results of pointersect and baselines on Sketchfab dataset. Mesh credit: 🅭 Marchal [20]. 🅭 Marchal [19].

In Fig. 16 we show an additional result where Poisson reconstruction is given the *estimated* vertex normal. We estimate the vertex normal directly from the point cloud using Open3D, which estimates vertex normal by fitting local planes. As can be seen, the output quality of Possion reconstruction is significantly affected when we use vertex normal that contains a small amount of noise. Pointersect, on the other hand, does not use vertex normal, so the result is unaffected.

I. Noisy point cloud from handheld devices

Pointersect is trained on *clean* point clouds that are created from meshes. Thus, pointersect renders the input point cloud as is—if an input point cloud contains noisy points, the noise will appear in the rendered images and estimated depth and normal maps. It would be interesting to see what would happen if we directly apply the pointersect model on a noisy point cloud that is captured by a handheld device.

We take 19 RGBD images from the ARKitScenes dataset [4], where the depth maps and camera poses are estimated by ARKit. We perform a simple point-cloud outlier removal utilizing the confidence map output by ARKit. We then perform a voxel downsampling on the noisy point cloud with a cell width of 0.05. We apply various methods to render novel views, including visibility splatting where each point is rendered as 1 pixel, screened Poisson reconstruction,

Table 7. Test results on three datasets. Inputs are 30 or 60 RGBD images captured at random locations within sphere shell, pointing toward the center. All test meshes are unseen during training. NGP does not use depth information and is trained for 1000 epochs (about 10 minutes). It is included as a reference baseline.

Method	Metrics	tex-models		ShapeNet		Sketchfab	
		30 input views	60 input views	30 input views	60 input views	30 input views	60 input views
Visibility splatting	depth (RMSE) ↓	0.06 ± 0.07	0.04 ± 0.03	0.03 ± 0.03	0.03 ± 0.03	0.05 ± 0.03	0.04 ± 0.02
	normal (angle (°)) ↓	5.59 ± 2.40	5.72 ± 2.64	$\mathbf{6.85 \pm 3.67}$	$\mathbf{7.01 \pm 3.82}$	7.84 ± 3.06	8.13 ± 3.31
	hit (accuracy (%)) ↑	99.1 ± 0.3	99.0 ± 0.3	99.1 ± 0.5	99.1 ± 0.6	99.3 ± 0.2	99.2 ± 0.2
	color (PSNR (dB)) ↑	22.0 ± 1.8	21.9 ± 2.1	24.0 ± 3.0	23.8 ± 3.1	23.3 ± 1.3	23.2 ± 1.5
	color (SSIM) ↑	0.9 ± 0.0	0.9 ± 0.0	0.9 ± 0.1	0.9 ± 0.1	0.9 ± 0.1	0.9 ± 0.1
	color (LPIPS) ↓	0.07 ± 0.05	0.05 ± 0.02	0.05 ± 0.03	0.04 ± 0.03	0.08 ± 0.06	0.07 ± 0.03
Poisson surface recon.	depth (RMSE) ↓	0.02 ± 0.04	0.02 ± 0.04	0.03 ± 0.07	0.03 ± 0.06	0.06 ± 0.09	0.06 ± 0.09
	normal (angle (°)) ↓	5.76 ± 2.65	5.46 ± 2.53	11.14 ± 6.04	10.49 ± 6.89	10.59 ± 6.51	10.14 ± 6.18
	hit (accuracy (%)) ↑	$\mathbf{99.9 \pm 0.1}$	$\mathbf{99.9 \pm 0.1}$	98.1 ± 7.3	99.3 ± 1.3	99.6 ± 0.4	99.6 ± 0.6
	color (PSNR (dB)) ↑	27.5 ± 3.1	27.7 ± 3.1	-	-	26.5 ± 3.6	26.5 ± 3.9
	color (SSIM) ↑	0.9 ± 0.0	0.9 ± 0.0	0.9 ± 0.1	0.9 ± 0.0	0.9 ± 0.0	0.9 ± 0.0
	color (LPIPS) ↓	0.05 ± 0.03	0.05 ± 0.03	0.05 ± 0.06	0.05 ± 0.04	0.08 ± 0.04	0.08 ± 0.04
Neural points [10]	depth (RMSE) ↓	0.04 ± 0.02	0.04 ± 0.02	0.03 ± 0.03	0.03 ± 0.03	0.04 ± 0.02	0.04 ± 0.02
	normal (angle (°)) ↓	12.59 ± 2.73	13.00 ± 2.58	16.93 ± 4.57	16.86 ± 4.44	15.24 ± 3.21	15.64 ± 3.15
	hit (accuracy (%)) ↑	99.0 ± 0.4	99.0 ± 0.4	99.0 ± 0.7	99.0 ± 0.7	99.2 ± 0.2	99.2 ± 0.2
	color (PSNR (dB)) ↑	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
	color (SSIM) ↑	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
	color (LPIPS) ↓	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
NPBG++ [31]	depth (RMSE) ↓	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
	normal (angle (°)) ↓	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
	hit (accuracy (%)) ↑	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
	color (PSNR (dB)) ↑	17.2 ± 2.4	17.3 ± 2.4	19.8 ± 4.1	19.8 ± 4.1	18.8 ± 1.7	18.9 ± 1.7
	color (SSIM) ↑	0.7 ± 0.1	0.7 ± 0.1	0.8 ± 0.1	0.8 ± 0.1	0.8 ± 0.1	0.8 ± 0.1
	color (LPIPS) ↓	0.23 ± 0.05	0.23 ± 0.05	0.17 ± 0.08	0.17 ± 0.08	0.21 ± 0.07	0.21 ± 0.06
NGP [24]	depth (RMSE) ↓	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
	normal (angle (°)) ↓	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
	hit (accuracy (%)) ↑	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
	color (PSNR (dB)) ↑	18.3 ± 7.7	22.3 ± 8.2	-	-	23.7 ± 6.9	24.8 ± 7.3
	color (SSIM) ↑	0.8 ± 0.1	0.8 ± 0.2	0.9 ± 0.1	0.9 ± 0.1	0.9 ± 0.1	0.9 ± 0.1
	color (LPIPS) ↓	0.22 ± 0.14	0.16 ± 0.19	0.14 ± 0.10	0.17 ± 0.14	0.14 ± 0.10	0.12 ± 0.11
Proposed ($k = 40$)	depth (RMSE) ↓	0.02 ± 0.02	0.02 ± 0.02	0.02 ± 0.02	0.02 ± 0.02	0.03 ± 0.02	0.03 ± 0.02
	normal (angle (°)) ↓	5.85 ± 1.82	6.30 ± 1.75	8.85 ± 3.61	9.25 ± 3.29	7.01 ± 2.11	7.47 ± 2.02
	hit (accuracy (%)) ↑	$\mathbf{99.9 \pm 0.0}$	$\mathbf{99.9 \pm 0.0}$	$\mathbf{99.8 \pm 0.3}$	$\mathbf{99.8 \pm 0.2}$	$\mathbf{99.9 \pm 0.0}$	$\mathbf{99.9 \pm 0.0}$
	color (PSNR (dB)) ↑	$\mathbf{32.4 \pm 2.1}$	$\mathbf{33.3 \pm 1.9}$	$\mathbf{29.7 \pm 3.8}$	$\mathbf{30.5 \pm 3.9}$	30.3 ± 2.7	30.9 ± 2.7
	color (SSIM) ↑	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$
	color (LPIPS) ↓	0.02 ± 0.02	$\mathbf{0.01 \pm 0.02}$	$\mathbf{0.03 \pm 0.03}$	$\mathbf{0.02 \pm 0.02}$	$\mathbf{0.04 \pm 0.04}$	$\mathbf{0.04 \pm 0.04}$
Proposed ($k = 200$)	depth (RMSE) ↓	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$
	normal (angle (°)) ↓	$\mathbf{5.09 \pm 1.83}$	$\mathbf{4.91 \pm 1.65}$	8.08 ± 3.57	7.59 ± 3.38	$\mathbf{5.88 \pm 1.96}$	$\mathbf{5.43 \pm 1.69}$
	hit (accuracy (%)) ↑	$\mathbf{99.9 \pm 0.1}$	$\mathbf{99.9 \pm 0.0}$	99.7 ± 0.3	$\mathbf{99.8 \pm 0.3}$	$\mathbf{99.9 \pm 0.1}$	$\mathbf{99.9 \pm 0.0}$
	color (PSNR (dB)) ↑	$\mathbf{32.4 \pm 2.6}$	$\mathbf{33.3 \pm 2.6}$	$\mathbf{29.7 \pm 3.6}$	30.4 ± 3.8	$\mathbf{30.9 \pm 2.7}$	$\mathbf{31.6 \pm 2.8}$
	color (SSIM) ↑	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$
	color (LPIPS) ↓	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.01 \pm 0.00}$	$\mathbf{0.03 \pm 0.03}$	$\mathbf{0.02 \pm 0.02}$	$\mathbf{0.04 \pm 0.04}$	$\mathbf{0.04 \pm 0.04}$

IBRNet [43], NGP [24] where we train the model for 200 epochs, and the same pointersect model used in the paper that is trained on clean point clouds. We use the same setting as those in Section 4.6: $k = 100, \delta = 0.2$. Note that Poisson and NGP utilize per-scene optimization; IBRNet does not utilize depth information; pointersect never sees real noisy RGBD images during training. It is simply an exploratory experiment to inspire future work.

Figure 15 shows the results. As can be seen, while the pointersect model never sees real-world noisy point clouds during training, it is able to render the point cloud with reasonable quality. Utilizing the depth information, pointersect can directly renders the input images without any training, and the rendered results do not contain floating

Table 8. Test results on the Stanford Bunny mesh with a single input RGBD image captured at the frontal view. Target views are a frontal circle looking towards the mesh.

Method	Metrics	1 view 25 × 25	1 view 50 × 50	1 view 100 × 100	1 view 200 × 200	5 views 25 × 25	5 views 50 × 50	5 views 100 × 100	5 views 200 × 200
Visibility splatting	depth (RMSE) ↓	0.07 ± 0.05	0.07 ± 0.04	0.05 ± 0.03	0.02 ± 0.01	0.21 ± 0.02	0.19 ± 0.02	0.15 ± 0.01	0.04 ± 0.01
	normal (angle (°)) ↓	4.24 ± 0.44	4.28 ± 0.34	3.82 ± 0.26	3.07 ± 0.10	11.53 ± 0.87	10.39 ± 0.82	7.62 ± 0.56	3.64 ± 0.22
	hit (accuracy (%)) ↑	62.8 ± 1.2	64.9 ± 1.2	73.1 ± 1.1	93.0 ± 1.0	63.5 ± 1.2	67.5 ± 1.2	79.7 ± 0.9	97.7 ± 0.1
	color (PSNR (dB)) ↑	11.0 ± 0.2	11.2 ± 0.2	12.4 ± 0.3	18.2 ± 0.3	11.1 ± 0.2	11.5 ± 0.2	13.5 ± 0.3	21.6 ± 0.2
	color (SSIM) ↑	0.6 ± 0.0	0.6 ± 0.0	0.6 ± 0.0	0.8 ± 0.0	0.6 ± 0.0	0.6 ± 0.0	0.6 ± 0.0	0.8 ± 0.0
	color (LPIPS) ↓	0.42 ± 0.02	0.36 ± 0.02	0.33 ± 0.01	0.17 ± 0.01	0.38 ± 0.02	0.32 ± 0.01	0.29 ± 0.01	0.12 ± 0.01
Poisson surface recon.	depth (RMSE) ↓	0.09 ± 0.02	0.09 ± 0.03	0.08 ± 0.03	0.07 ± 0.03	0.07 ± 0.02	0.05 ± 0.02	0.03 ± 0.02	0.02 ± 0.02
	normal (angle (°)) ↓	27.45 ± 1.45	19.96 ± 1.28	13.44 ± 1.67	8.79 ± 1.53	21.46 ± 1.07	14.90 ± 1.00	8.69 ± 0.69	5.11 ± 0.44
	hit (accuracy (%)) ↑	76.3 ± 3.3	72.2 ± 4.8	72.2 ± 4.8	73.0 ± 4.9	76.6 ± 3.2	77.7 ± 4.5	82.4 ± 3.9	88.3 ± 1.2
	color (PSNR (dB)) ↑	12.0 ± 0.6	10.6 ± 0.8	11.0 ± 0.8	11.2 ± 0.9	11.5 ± 0.6	12.1 ± 1.2	13.4 ± 1.5	15.7 ± 0.6
	color (SSIM) ↑	0.4 ± 0.0	0.4 ± 0.0	0.5 ± 0.0	0.6 ± 0.0	0.4 ± 0.0	0.5 ± 0.0	0.6 ± 0.0	0.8 ± 0.0
	color (LPIPS) ↓	0.53 ± 0.03	0.51 ± 0.04	0.45 ± 0.03	0.38 ± 0.03	0.48 ± 0.03	0.41 ± 0.04	0.30 ± 0.04	0.19 ± 0.02
Neural points [10]	depth (RMSE) ↓	0.04 ± 0.01	0.03 ± 0.01	0.02 ± 0.00	0.02 ± 0.00	0.07 ± 0.01	0.04 ± 0.01	0.06 ± 0.01	0.04 ± 0.01
	normal (angle (°)) ↓	18.01 ± 0.14	14.94 ± 0.41	11.83 ± 0.50	9.73 ± 0.31	18.37 ± 0.36	15.04 ± 0.78	11.91 ± 0.68	9.87 ± 0.62
	hit (accuracy (%)) ↑	86.9 ± 1.2	95.1 ± 1.2	96.3 ± 1.2	96.8 ± 1.2	94.4 ± 0.7	98.1 ± 0.1	98.3 ± 0.2	98.7 ± 0.1
	color (PSNR (dB)) ↑	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
	color (SSIM) ↑	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
	color (LPIPS) ↓	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.	not supp.
Proposed	depth (RMSE) ↓	0.02 ± 0.01	0.02 ± 0.00	0.01 ± 0.01	0.01 ± 0.00	0.03 ± 0.01	0.02 ± 0.01	0.01 ± 0.01	0.01 ± 0.01
	normal (angle (°)) ↓	15.58 ± 0.45	8.73 ± 0.31	5.07 ± 0.10	3.90 ± 0.10	15.07 ± 0.69	7.79 ± 0.52	4.87 ± 0.26	3.86 ± 0.16
	hit (accuracy (%)) ↑	91.4 ± 1.0	95.4 ± 1.2	96.4 ± 1.2	96.8 ± 1.2	96.3 ± 1.0	99.4 ± 0.1	99.7 ± 0.1	99.8 ± 0.1
	color (PSNR (dB)) ↑	16.4 ± 0.4	19.2 ± 0.7	20.6 ± 1.0	21.8 ± 1.4	18.7 ± 0.9	22.9 ± 0.4	25.5 ± 0.3	28.8 ± 0.6
	color (SSIM) ↑	0.6 ± 0.0	0.7 ± 0.0	0.8 ± 0.0	0.9 ± 0.0	0.7 ± 0.0	0.8 ± 0.0	0.9 ± 0.0	0.9 ± 0.0
	color (LPIPS) ↓	0.32 ± 0.01	0.26 ± 0.01	0.16 ± 0.01	0.09 ± 0.01	0.29 ± 0.01	0.20 ± 0.01	0.10 ± 0.01	0.04 ± 0.00

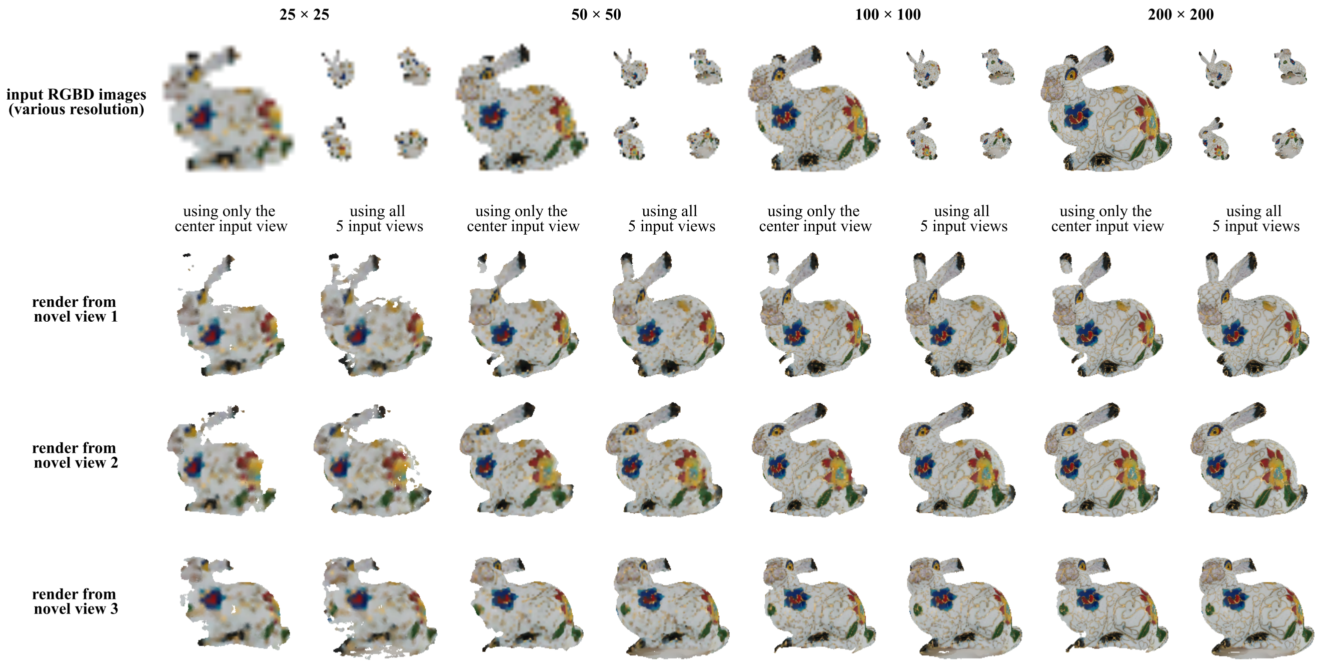


Figure 14. The effect of number of views and the sampling rate of the input point cloud. We create various kinds of input point clouds of the Stanford Bunny [49] by capturing 1 or 5 input RGBD images of various resolutions. The first row shows the 5 input RGBD images (with the center one enlarged). The input point cloud becomes denser from left to right when the resolution of the input images increases. With 5 input views, the point cloud covers a wider area than using only the center input view. The bottom three rows show the output rendering from pointersect at 3 novel viewpoints.

artifacts. Since pointersect does not perform any per-scene optimization, the rendered images present the varying exposure and white-balance settings in the input images / point cloud. While the surface normal estimation are reasonable, they are affected most severe by the noise in the point cloud and camera poses compared to the estimated depth map and the results when the input point cloud is clean.

J. Sketchfab dataset

We train our model on a subset of the Sketchfab dataset that was used by Qian et al. [30] to train Neural Points [10]. The original dataset contains 90 training meshes and 13 test meshes. However, we use only the 48 training meshes that are with variants of the Creative Common license and searchable on sketchfab.com. We use the same 13 test meshes — they are all with the Creative Common license. In Figure 17, we plot the training and test meshes, and in Table 9 and Table 10 we list their download links and license information.

References

[1] Anders Adamson and Marc Alexa. Approximating and intersecting surfaces from points. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 230–239, 2003. 2

[2] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *European Conference on Computer Vision*, pages 696–712. Springer, 2020. 2

[3] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. 1

[4] Gilad Baruch, Zhuoyuan Chen, Afshin Dehghan, Tal Dimry, Yuri Feigin, Peter Fu, Thomas Gebauer, Brandon Joffe, Daniel Kurz, Arik Schwartz, and Elad Shulman. ARKitScenes - a diverse real-world dataset for 3d indoor scene understanding using mobile RGB-D data. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. 7, 14

[5] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T Barron, Ce Liu, and Hendrik Lensch. Nerd: Neural reflectance decomposition from image collections. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12684–12694, 2021. 1, 2

[6] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14124–14133, 2021. 1, 2

[7] Inchang Choi, Orazio Gallo, Alejandro Troccoli, Min H Kim, and Jan Kautz. Extreme view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7781–7790, 2019. 1, 2

[8] Peng Dai, Yinda Zhang, Zhuwen Li, Shuaicheng Liu, and Bing Zeng. Neural point cloud rendering via multi-plane projection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7830–7839, 2020. 2

[9] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised nerf: Fewer views and faster training for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12882–12891, 2022. 1, 2

[10] Wanquan Feng, Jin Li, Hongrui Cai, Xiaonan Luo, and Juyong Zhang. Neural points: Point cloud representation with neural fields for arbitrary upsampling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18633–18642, 2022. 2, 8, 9, 10

[11] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022. 1, 2

[12] Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. Point2mesh: a self-prior for deformable meshes. *ACM Transactions on Graphics (TOG)*, 39(4):126–1, 2020. 2

[13] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 3

[14] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006. 2

[15] Petr Kellnhofer, Lars C Jebe, Andrew Jones, Ryan Spicer, Kari Pulli, and Gordon Wetzstein. Neural lumigraph rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4287–4297, 2021. 1, 2

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 4

[17] Christoph Lassner and Michael Zollhofer. Pulsar: Efficient sphere-based neural rendering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1440–1449, 2021. 1, 2

[18] Baorui Ma, Zhizhong Han, Yu-Shen Liu, and Matthias Zwicker. Neural-pull: Learning signed distance function from point clouds by learning to pull space onto surface. In *International Conference on Machine Learning (ICML)*, pages 7246–7257. PMLR, 2021. 2

[19] Geoffrey Marchal. Cupid. https://skfb.ly/6vN6Z. 7

[20] Geoffrey Marchal. Seated jew. https://skfb.ly/6ynCI. 7

[21] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019. 1, 2

[22] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 1, 2, 3

[23] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022.

Table 9. Download links and credits of the entire Sketchfab training set used to train the model.

Name	Download link	Credit
Alliance-statue	https://skfb.ly/DHIV	"Warcraft Alliance Statue" (https://skfb.ly/DHIV) by ahtiandr is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Amphitrite	https://skfb.ly/6n9PF	"Amphitrite - Louvre Museum (Low Definition)" (https://skfb.ly/6n9PF) by Benjamin Bardou is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Ancient-turti	https://skfb.ly/6o9KR	"Archelon - Dragon Sea Turtle" (https://skfb.ly/6o9KR) by inkhero is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Angel5	https://skfb.ly/6svWA	"Angel Sculpture 3D Scan (Einscan-S)" (https://skfb.ly/6svWA) by 3DWP is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Angel6	https://skfb.ly/WyIS	"Angels" (https://skfb.ly/WyIS) by rvscanners is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Angel-statue	https://skfb.ly/6qxUQ	"Angel Statue in Fossano" (https://skfb.ly/6qxUQ) by Albyfos is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Angel2	https://skfb.ly/SNow	"Angel 00" (https://skfb.ly/SNow) by TomaszGap is licensed under CC Attribution-NonCommercial-NoDerivs (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Angel3	https://skfb.ly/KVRY	"Angel" (https://skfb.ly/KVRY) by Medolino is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Angel4	https://skfb.ly/SZOv	"Baptismal Angel kneeling" (https://skfb.ly/SZOv) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Angel-diffuse2	https://skfb.ly/69usM	"Angel playing harp" (https://skfb.ly/69usM) by OpenScan is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Armadillo	https://skfb.ly/otzQs	"Stanford Armadillo PBR" (https://skfb.ly/otzQs) by hackmans is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Buddha-sit	https://skfb.ly/6nXwQ	"Buddha" (https://skfb.ly/6nXwQ) by icenvain is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Camera	https://skfb.ly/Lp7U	"Panasonic GH4 Body" (https://skfb.ly/Lp7U) by ScanSource 3D is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Compressor	https://skfb.ly/6oIZS	"Compressor Scan" (https://skfb.ly/6oIZS) by GoMeasure3D is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Dragon-plate	https://skfb.ly/osUnp	"Dragon (decimated sculpt)" (https://skfb.ly/osUnp) by Ashraf Bouhadida is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Dragon- stand	https://skfb.ly/6oTOQ	"Chinese Dragon" (https://skfb.ly/6oTOQ) by icenvain is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Dragon-warrior	https://skfb.ly/otvrO	"Dragon-stl" (https://skfb.ly/otvrO) by Thunk3D 3D Scanner is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Dragon-wing	https://skfb.ly/M9KW	"Wooden Dragon" (https://skfb.ly/M9KW) by jschmidtcreaform is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Dragon2	https://skfb.ly/BZsM	"XYZ RGB Dragon" (https://skfb.ly/BZsM) by 3D graphics 101 is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Fox-skull	https://skfb.ly/6UoqE	"Grey Fox skull" (https://skfb.ly/6UoqE) by RISD Nature Lab is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Ganesha	https://skfb.ly/66opT	"The elephant god Ganesha" (https://skfb.ly/66opT) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Ganesha-plane	https://skfb.ly/ovrr6	"Ganesha" (https://skfb.ly/ovrr6) by Paulotronics is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Gargo	https://skfb.ly/6S8qQ	"Gargo" (https://skfb.ly/6S8qQ) by rudyprieto is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Grid-dog	https://skfb.ly/DTuH	"Girl With Dog" (https://skfb.ly/DTuH) by pencas is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Golden-elephant	https://skfb.ly/GZSL	"Golden Ephant" (https://skfb.ly/GZSL) by dievitacola is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Guanyiny	https://skfb.ly/ZuHU	"Guanyin" (https://skfb.ly/ZuHU) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Happy-vrip	https://skfb.ly/BYQD	"Happy Buddha (Stanford)" (https://skfb.ly/BYQD) by 3D graphics 101 is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Lion-ball	https://skfb.ly/6GDqU	"King of Narnia: ASLAN" (https://skfb.ly/6GDqU) by Anahit Takiryan is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Man-face	https://skfb.ly/GrGu	"Bust of a Roman" (https://skfb.ly/GrGu) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Man-statue	https://skfb.ly/T9oz	"Man On Bench Statue PHOTOGRAMMETRY" (https://skfb.ly/T9oz) by MrDavids1 is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Maria	https://skfb.ly/6tzH7	"Maria Fjodorovna Barjatinskaja" (https://skfb.ly/6tzH7) by Geoffrey Marchal is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Mesh-little-angle	https://skfb.ly/6s8yO	"Little Angle" (https://skfb.ly/6s8yO) by MicMac is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Modello-buddha	https://skfb.ly/6qnIn	"Wooden Buddha statuette" (https://skfb.ly/6qnIn) by andrea.notarstefano is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Mozart	https://skfb.ly/FZyK	"The Infant Mozart" (https://skfb.ly/FZyK) by Geoffrey Marchal is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Roman- sphinx	https://skfb.ly/Y9sx	"Roman Sphinx" (https://skfb.ly/Y9sx) by tony-eight is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Snake	https://skfb.ly/BrOL	"Cobra Statue" (https://skfb.ly/BrOL) by Jonathan Williamson is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Statue-air-force	https://skfb.ly/LZBS	"a sculpture in Air Force Museum of Vietnam" (https://skfb.ly/LZBS) by HoangHiepVu is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Statue-child-fish	https://skfb.ly/6pJ9s	"Château de Chamarande - France" (https://skfb.ly/6pJ9s) by Sakado is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Statue-death	https://skfb.ly/ovIRF	"The death and the mother" (https://skfb.ly/ovIRF) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Statue-madona	https://skfb.ly/Lv9v	"Madona Sculpture" (https://skfb.ly/Lv9v) by jan.zachar is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Statue-mother	https://skfb.ly/6pUOp	"Pieta" (https://skfb.ly/6pUOp) by MSU Broad Art Museum is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Statue-napoleon	https://skfb.ly/6xHwD	"Equestrian statue of Napoleon" (https://skfb.ly/6xHwD) by Lo¨ic Norgeot is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Statue- neptune-horse	https://skfb.ly/6npKK	"Neptune - Louvre Museum" (https://skfb.ly/6npKK) by Benjamin Bardou is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Statue-oxen	https://skfb.ly/R9Ps	"Ox Statue (Kek Lok Si Buddhist Temple, Penang)" (https://skfb.ly/R9Ps) by nate_siddle is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Two-wrestiers-in-combat	https://skfb.ly/RTzv	"Two wrestlers in combat (repost)" (https://skfb.ly/RTzv) by Geoffrey Marchal is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
Vase-empire	https://skfb.ly/6uDFR	"Empire vase" (https://skfb.ly/6uDFR) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Vishnu	https://skfb.ly/6nCoN	"The God Vishnu" (https://skfb.ly/6nCoN) by Geoffrey Marchal is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

Table 10. Download links and credits of the test set of the Sketchfab dataset [30].

Name	Download link	Credit
A9-vulcan_aligned	https://skfb.ly/6AAGM	”Vulcan” (https://skfb.ly/6AAGM) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
a72-seated_jew_aligned	https://skfb.ly/6ynCI	”Seated Jew” (https://skfb.ly/6ynCI) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
asklepios_aligned	https://skfb.ly/6zt76	”Young roman as Asklepios” (https://skfb.ly/6zt76) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
baron_seutin_aligned	https://skfb.ly/6Bq6Y	”Baron Seutin” (https://skfb.ly/6Bq6Y) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
charite_-_CleanUp_-_LowPoly_aligned	https://skfb.ly/6yWCX	”The Charity” (https://skfb.ly/6yWCX) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
cheval_terracotta_-_LowPoly-RealOne_aligned	https://skfb.ly/6APDT	”Relief in terracotta” (https://skfb.ly/6APDT) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
cupid_aligned	https://skfb.ly/6vN6Z	”Cupid” (https://skfb.ly/6vN6Z) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
dame_assise_-_CleanUp_-_LowPoly_aligned	https://skfb.ly/6BNFn	”Seated Lady” (https://skfb.ly/6BNFn) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
drunkard_-_CleanUp_-_LowPoly_aligned	https://skfb.ly/6BH8D	”Drunkard - Cap Re” (https://skfb.ly/6BH8D) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
Gramme_aligned	https://skfb.ly/6ABqV	”Statue of Zénobe Gramme” (https://skfb.ly/6ABqV) by LZ Creation is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
madeleine_aligned	https://skfb.ly/6AtTy	”Marie-Madeleine” (https://skfb.ly/6AtTy) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
retheur_-_LowPoly_aligned	https://skfb.ly/ooIRB	”Bust of a rhetorician_Restored” (https://skfb.ly/ooIRB) by Digital_Restoration is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).
saint_lambert_aligned	https://skfb.ly/6ANWw	”Lambert de Maestricht and Liège” (https://skfb.ly/6ANWw) by Geoffrey Marchal is licensed under Creative Commons Attribution-NonCommercial (http://creativecommons.org/licenses/by-nc/4.0/).

1, 2

[24] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (TOG)*, 41(4):102:1–102:15, July 2022. 8

[25] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020. 1, 2

[26] Julian Ost, Issam Laradji, Alejandro Newell, Yuval Bahat, and Felix Heide. Neural point light fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18419–18429, 2022. 2

[27] Songyou Peng, Chiyu Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, and Andreas Geiger. Shape as points: A differentiable poisson solver. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:13032–13044, 2021. 2

[28] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, 2000. 2

[29] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016. 1

[30] Yue Qian, Junhui Hou, Sam Kwong, and Ying He. Pugeo-net: A geometry-centric network for 3d point cloud upsampling. In *European Conference on Computer Vision (ECCV)*, pages 752–769. Springer, 2020. 3, 10, 12, 15

[31] Ruslan Rakhimov, Andrei-Timotei Ardelean, Victor Lempitsky, and Evgeny Burnaev. NPBG++: Accelerating neural point-based graphics. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15969–15979, 2022. 2, 8

[32] Prajit Ramachandran, Barret Zoph, and Quoc Le. Searching for activation functions. In *International Conference on Learning Representations (ICLR)*, 2018. 3

[33] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021. 1, 2

[34] Gernot Riegler and Vladlen Koltun. Free view synthesis. In *European Conference on Computer Vision*, pages 623–640. Springer, 2020. 1, 2

[35] Gernot Riegler and Vladlen Koltun. Stable view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12216–12225, 2021. 1, 2

[36] Barbara Roessle, Jonathan T Barron, Ben Mildenhall, Pratul P Srinivasan, and Matthias Nießner. Dense depth priors for neural radiance fields from sparse input views. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12892–12901, 2022. 1, 2

[37] Darius Rückert, Linus Franke, and Marc Stamminger. ADOP: Approximate differentiable one-pixel point rendering. *ACM Transactions on Graphics (TOG)*, 41(4):1–14, 2022. 2

[38] Vincent Sitzmann, Semon Rezchikov, Bill Freeman, Josh Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. *Advances in Neural Information Processing Systems*, 34:19313–19325, 2021. 1, 2

[39] Pratul P Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T Barron. Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7495–7504, 2021. 1, 2

[40] Mohammed Suhail, Carlos Esteves, Leonid Sigal, and Ameesh Makadia. Light field neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8269–8279, 2022. 1, 2

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, 30:5998–6008, 2017. 3, 4

[42] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 1

[43] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2021. 1, 2, 8

[44] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5438–5448, 2022. 2

[45] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33:2492–2502, 2020. 1, 2

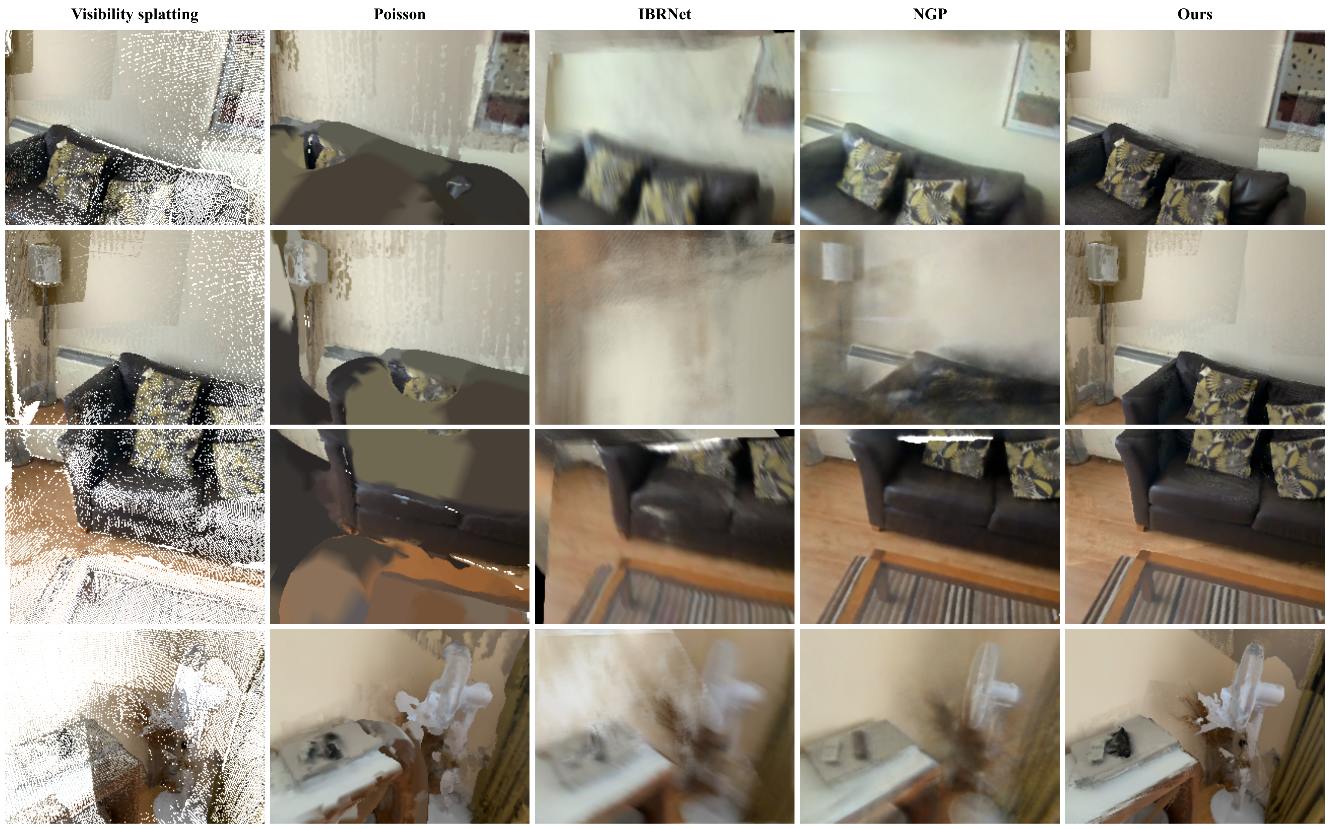
[46] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021. 1, 2

[47] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021. 1, 2

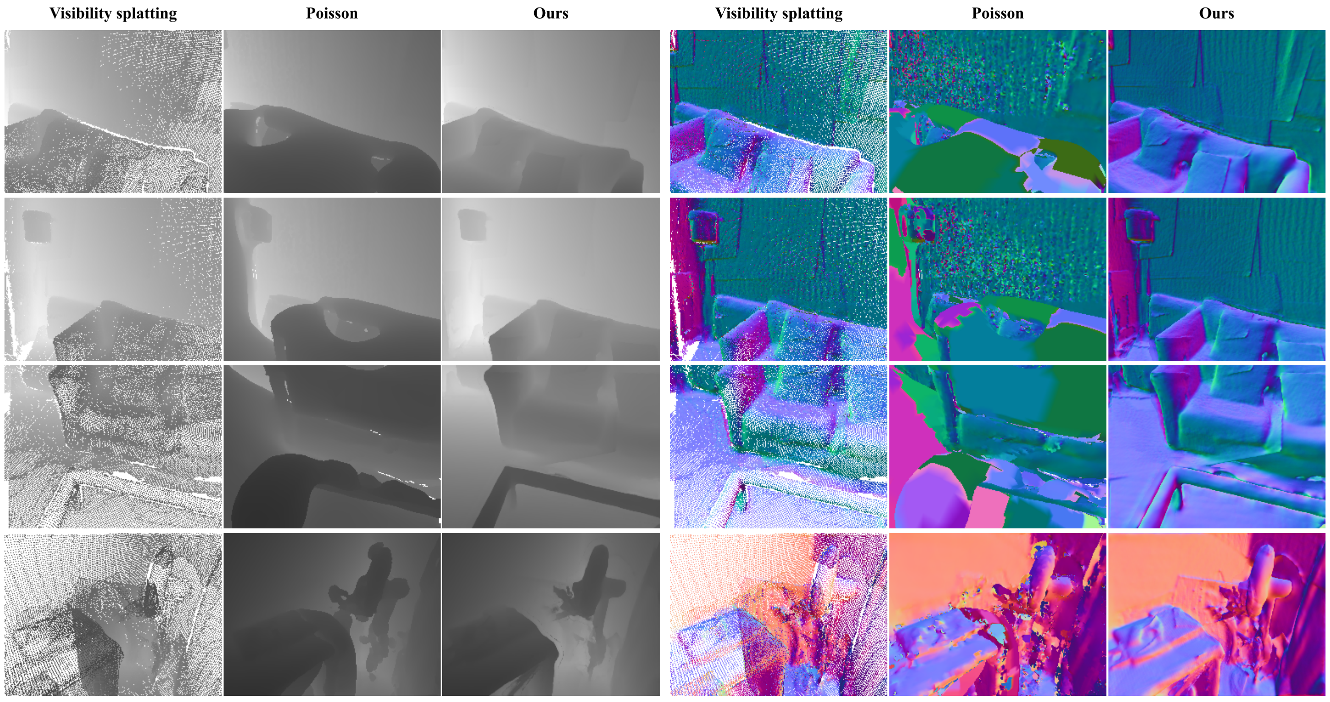
[48] Xiuming Zhang, Pratul P Srinivasan, Boyang Deng, Paul Debevec, William T Freeman, and Jonathan T Barron. Nerfactor: Neural factorization of shape and reflectance under an unknown illumination. *ACM Transactions on Graphics (TOG)*, 40(6):1–18, 2021. 1, 2

[49] Kun Zhou, Xi Wang, Yiying Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. Texturemontage: Seamless texturing of arbitrary surfaces from multiple images. *ACM Transactions on Graphics (TOG)*, 24(3):1148–1155, 2005. Available at http://kunzhou.net/tex-models.htm. 5, 6, 9

[50] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 4



(a) Novel-view rendering on noisy point cloud captured by a handheld device



(b) Depth (left) and surface normal (right) estimation at the viewpoints in (a).

Figure 15. Results on noisy depth maps captured by a handheld device. Inputs are 19 RGBD images from the ARKitScenes dataset [4], where the depth maps and camera poses are estimated by ARKit and thus contain noise. Note that the pointersect model is not trained to render noisy point clouds and has never seen real noisy RGBD images during training. The result is provided to motivate future work.

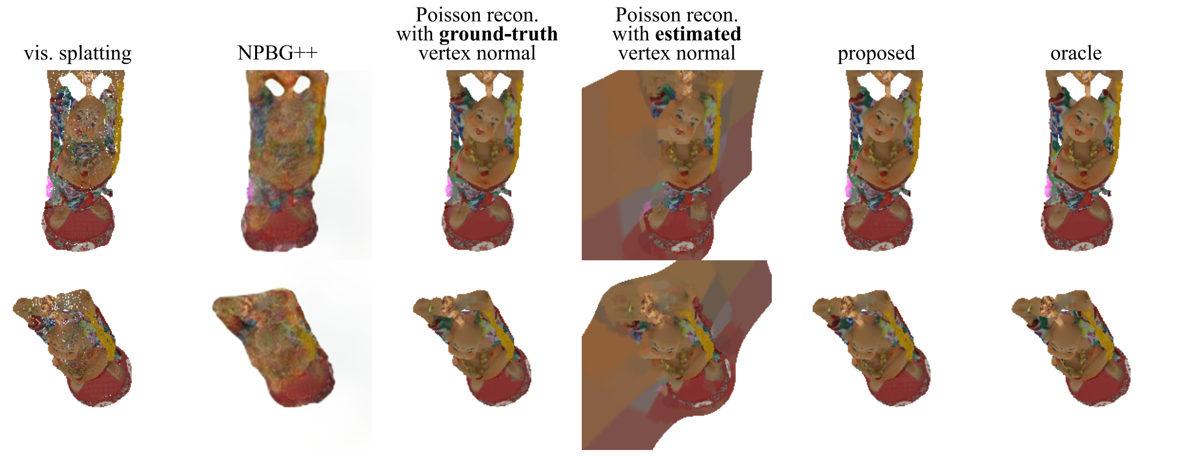

vis. splatting
NPBG++
Poisson recon. with **ground-truth** vertex normal
Poisson recon. with **estimated** vertex normal
proposed
oracle

Figure 16. Novel view rendering on Buddha. The output quality of Poisson surface reconstruction depends highly on the quality of the vertex normal. Pointersect does not need nor use vertex normal.

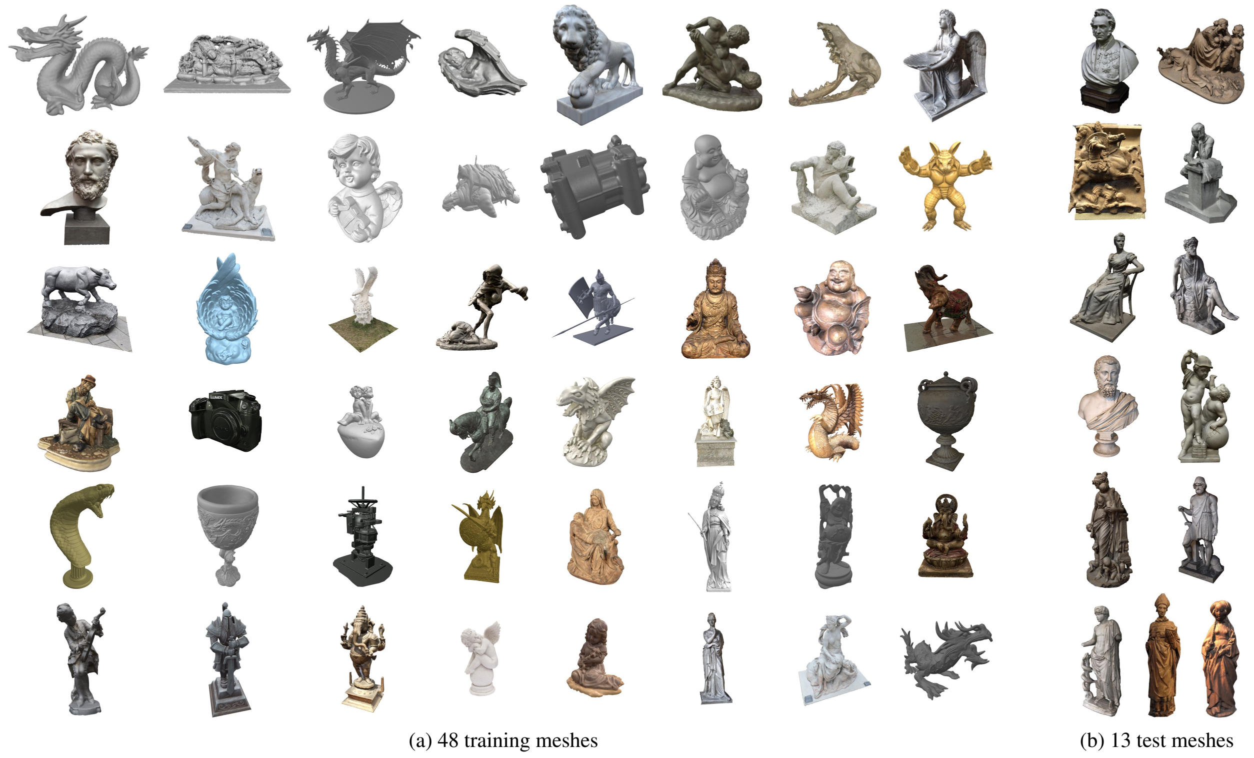


Figure 17. (a) The entire training meshes used to train our model. The meshes are from a subset of the Sketchfab dataset [30]. (b) is the test meshes. See Table 9 and Table 10 for credits.