Neural Voxel Renderer: Learning an Accurate and Controllable Rendering Tool

Konstantinos Rematas Vittorio Ferrari
Google Research

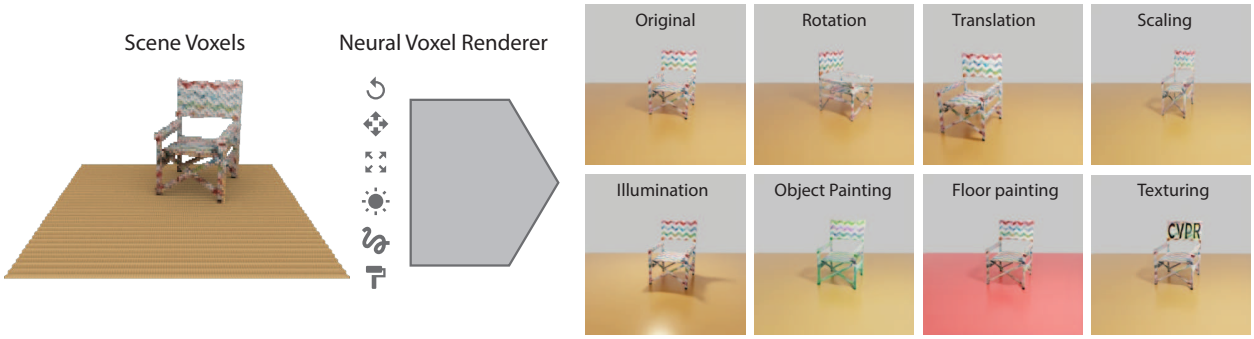


Figure 1. Neural voxel renderer converts a set of colored voxels into a realistic and detailed image. It also allows elaborate modifications in the geometry or the appearance of the input that are faithfully represented in the synthesized image.

Abstract

We present a neural rendering framework that maps a voxelized scene into a high quality image. Highly-textured objects and scene element interactions are realistically rendered by our method, despite having a rough representation as an input. Moreover, our approach allows controllable rendering: geometric and appearance modifications in the input are accurately propagated to the output. The user can move, rotate and scale an object, change its appearance and texture or modify the position of the light and all these edits are represented in the final rendering. We demonstrate the effectiveness of our approach by rendering scenes with varying appearance, from single color per object to complex, high-frequency textures. We show that our rerendering network can generate very detailed images that represent precisely the appearance of the input scene. Our experiments illustrate that our approach achieves more accurate image synthesis results compared to alternatives and can also handle low voxel grid resolutions. Finally, we show how our neural rendering framework can capture and faithfully render objects from real images and from a diverse set of classes.

1. Introduction

What is the typical process for rendering a synthetic scene? In a 3D graphics software, like Blender [3] and 3D Studio Max [2], the user creates a set of geometric objects in a virtual 3D world, edits their material properties and adds the light sources. Once the desired configuration is achieved, the program renders the 3D scene into an image using a rendering algorithm such as Path Tracing [24]. While this setup unfolds the creativity of the user, it increases the learning complexity of the system, requires a lot of manual input and it is not differentiable.

The emergence of deep generative models introduced a new image synthesis medium. Generative adversarial networks [18] are able to produce highly realistic images of faces [25] or Imagenet [57] categories [9] using only class labels as input. Moreover, image-to-image translation [23] presented a framework where an input image featuring only partial information (*e.g.* only edges), can be transformed into a naturally looking one. Neural networks are also applied for graphics applications. Feed forward networks can learn a mapping between geometric attributes [44] or voxels [45] to shaded outputs, and rerendering networks can correct the artifacts of traditional rendering approaches [43, 41]. However, these approaches often produce blurry results and have limited control over the input: appearance changes are mostly restricted to viewpoint or high-level attributes.

In this paper we present a neural network model that learns how to render a scene given voxels as input. The scene can be modified in terms of appearance, location, orientation and lighting, and all changes are faithfully expressed in the rendered output. With only requiring a rough specification of the geometry as a voxel grid, our framework produces an accurate image, with plausible light interactions between the scene elements (*e.g.* casting shad-

ows, reflections, *etc*.). Our method includes a rerendering module which enables us to render highly textured objects precisely and in detail. Moreover, our framework naturally inputs limited appearance information. Instead of manually painting materials on the geometry [1] or requiring multiple views of the object [59, 6], our method can use a single image aligned with a 3D object to capture detailed appearance properties and propagate them to other views.

We demonstrate the ability of our approach to render realistic images in a comprehensive set of experiments. We show geometric and appearance modifications in synthetic datasets with increasing texture complexity and we reproduce the look of objects from real images as well. We illustrate how the network reproduces the interactions between the scene elements, *e.g*. specular reflections, shadows, secondary bounces of light. Finally, we compare our proposed framework with alternative approaches [45, 23, 64] and we attain better performance in several metrics. Our contributions can be summarized as:

- A neural rendering framework with controllable object appearance and scene illumination effects.
- Capturing texture details with a neural rerendering module.
- Learnable interactions between scene entities such as reflections, shadows and secondary bounces of light.

2. Related Work

Geometry-based neural rendering. One approach for image synthesis with geometric information is to replace the traditional rendering pipeline with a neural network. RenderNet [45] learns how to map a voxel grid to a shaded output, such as Phong shading. The method can also be used for normal estimation, allowing the use of the Phong illumination model [52], and generating textures of faces using PCA coefficients as input. Texture fields [47] is estimating the appearance of a 3D object using a function that maps a point in space to a color, conditioned on an input image. In contrast, our approach allows detailed manipulation and rendering of arbitrary textures and provides an adjustable illumination source (area light with soft shadows).

Deferred Neural Rendering [65] learns to synthesize novel views of a scene using neural textures, a learnable element that acts as a UV atlas. Deep Appearance Models [37] encode the facial geometry and texture of a particular person and can generate novel views during inference. Neural Volumes [38] focuses on the 3D reconstruction of an object by taking a set of calibrated views as an input and producing a 3D voxel grid that is rendered with differentiable ray-marching. While these approaches produce high quality results, their models are trained for a particular object/scene, limiting their applicability to general cases, and they assume static light. Also, these methods allow limited edits in the original scene as they focus on view synthesis.

Another direction is neural rerendering. Looking-good [41] uses a neural network to fix artifacts from a multiview capture and [43] rerenders a point cloud from a 3D reconstruction with modifiable appearance. Similarly, [53] estimate the RGB image from a structure-from-motion pointcloud. Deep Shading [44] converts a set of rendering buffers (position, normals, *etc*.) to shaded effects. Again, the ability to modify the output is bounded by either what it was presented during training or to holistic appearance transfers.

Image-based neural rendering. Given a set of images and their corresponding camera matrices, Deep Voxels [61] encode the view-dependent appearance of an object in a 3D latent voxel grid, which can be later used for rendering novel views. However, the latent voxels incorporate the appearance of a single object that undergoes viewpoint changes and the method requires a large number of calibrated images. Another generative approach is to synthesize the images directly given few attributes as an input [15] (*e.g*. transformation parameters, color *etc*.), but this approach needs access to the whole database of objects and the rendered images are often blurry.

The image-to-image translation [23] paradigm can also be seen as a form of neural rendering. Methods can synthesize new images of humans based on a 2D pose [35, 10, 40, 8] or convert semantic maps to natural images [67, 51]. Style transfer [17] can also alter the appearance of an input image in a realistic way [31, 39]. However, these approaches perform specific synthesis and editing tasks, where the control over the final rendering is based on the attributes from the supplied data.

Novel view synthesis is the task of generating a new view of an object given an input image from another view. This can be assisted by a 3D object [55], considering visible and hidden parts [71, 16] or directly from one [64] or multiple [48, 62] views. Again, these methods deal only with rotation modifications and their outputs often lose appearance details. An alternative to the previous explicit geometric transformations are the disentangled representations in feature space. The work of [27] learns to render different viewpoints and lighting conditions by a careful handling of the training procedure, while [69] considers the transformations directly on the latent features. Recently, [46] demonstrated the ability to generate realistic images in an unsupervised way using a disentangled latent volumetric representation. However, the modifications are again limited to rotations and simple lighting, while our network handles more complex appearance alterations explicitly on the input voxels.

Differentiable rendering. To overcome the non-differentiable nature of traditional rendering, previous works introduce a differentiable rasterizer [12, 36, 19] or splatting [70], propose a differentiable ray-tracer [66, 29] or have a differentiable, BRDF-based rendering model [14,

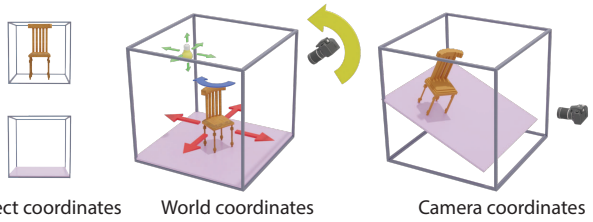


Figure 2. Scene setup: the object is placed in the world coordinates where it can be rotated and translated. The light can also be translated and the camera can change elevation. As network inputs, the scene is in camera coordinates and the light position is a 3-dimensional xyz vector.

32, 34, 30]. The focus of these methods is inverse rendering (estimate geometry, materials, *etc*.), while we are interested in the forward synthesis process.

3. Method

3.1. Overview

Our goal is to learn a network that renders a realistic image given a voxelized 3D input. While there are several types of 3D representations for neural rendering (*e.g*. meshes [19, 36], implicit functions [47, 49, 42], rendering buffers [44]), we choose to work with voxels because they are easy to input into a neural network and they provide the flexibility for arranging the scene elements in a natural way (*e.g*. a chair on top of the ground). The output of our network is a realistic image representing the scene from a particular viewpoint given by the user.

Scene definition. Our scene consists of three elements: the object, the ground and the light. The elements are placed in a bounded 3D world that is observed by a camera at a fixed distance. The object and the ground are represented as voxels and we use an area light as this generates more realistic soft shadows than a point light source.

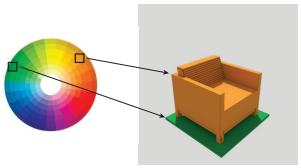
Our approach supports several editable attributes: (1) The object can be translated across the x and z axis and rotated around the y axis. Also, we can apply local rigid and non-rigid transformations to the voxels. (2) The light can be translated in a bounded volume above the ground. (3) The camera elevation can be adjusted. (4) The appearance of the object can change. For (4), we consider a spectrum of modifications, from painting the object/floor with a single uniform color to applying arbitrary textures to the object.

Apart from the editable parameters, the scene has some fixed attributes. There is ambient light, the color of the light is white, the camera focal length is fixed to 40, the object is diffuse, and the floor is slightly specular. Detailed values of the scene setup can be found in the supplementary material.

The network expects the scene to be in the camera coordinate frame. We first apply the modifications and then we convert the scene from world to camera coordinates.

3.2. Coloring Voxels

Manual coloring. A straightforward approach would be to color voxels manually. As a use case we consider the setting where the object and the floor have a single color each. The user can select the colors from a palette and assign them to the objects in the scene, similar to the bucket fill tool in most image editing software (see inset figure). The network still has to properly shade the scene based on the light position, and generate global illumination effects.



Colors from an image. Tools for coloring voxels can be found in programs like MagicaVoxel [5], but detailed voxel painting can be cumbersome. A more practical alternative is to perform image-based coloring. Given the alignment between the 3D object and an image depicting the object (we refer to that image as **appearance source** $\mathcal{A}$), we can un-project the color of the pixels directly onto the voxels. Alternatively, if the the input 3D object is in the camera space, we can assume orthographic projection. The centers of the voxels are projected to the image using the camera parameters. Finally, the color for every voxel is taken from the pixel it falls into. While this approach requires aligned images with the 3D objects, recent advantages in 2D to 3D alignment can provide such information automatically, see for example [54, 50, 7, 68, 22, 21, 20].

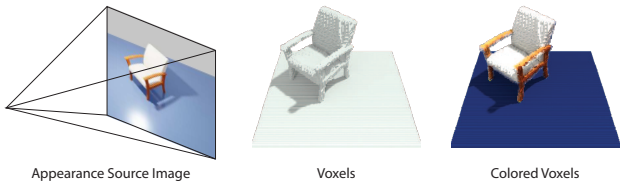


Figure 3. Coloring voxels from an image: given the 2D-to-3D alignment, the voxels take the color from the corresponding pixels of the appearance source image.

Appearance capture from a single image introduces artifacts typical with projective texturing. The assigned colors come from the input view and they contain view-dependent information such as shading, shadows, *etc*. Unlike typical image-based rendering, in our scenario the object will be rendered from a different viewpoint or lighting condition, so the rendered object colors need to change accordingly. We tackle this problem with a carefully prepared training set that includes this type of appearance changes (see Sec. 4.1).

Another aspect that requires attention is how to color the voxels that are not visible. We determine voxel visibility automatically using ray-marching [28]. A hidden voxel gets the color of the first visible voxel along its camera ray. We observed that this approach is beneficial in cases with thin

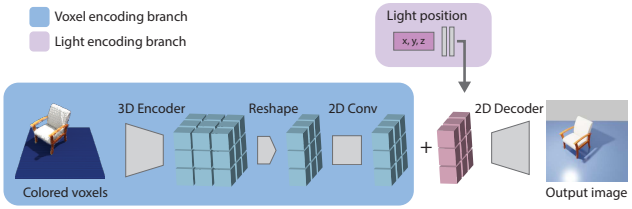


Figure 4. NVR network architecture. Two branches encode the voxel and light position inputs and a decoder combines their output and produces the final rendering.

structures (*e.g.* chair handles and legs), and the generated artifacts were well handled by the network.

Additionally, we take advantage of the symmetry in many man-made objects such as chairs and cars: if a voxel is not visible, we copy its color from its symmetric voxel across the y axis (if that is visible).

3.3. Neural Voxel Renderer (NVR)

Model architecture. Our Neural Voxel Renderer (NVR) network θ is illustrated in Fig. 4 (more details in the supplementary material). The inputs are (1) the scene represented as voxels $V \in \mathbb{R}^{128^3 \times 4}$ and (2) the light position $L \in \mathbb{R}^3$. The network output the image $I \in \mathbb{R}^{256^2 \times 3}$:

$$I = \theta(V, L) \tag{1}$$

The voxels V contain the RGB colors and visibility, which are automatically estimated based on the 2D-3D alignment (Sec. 3.2). We use RenderNet as our backbone [45]. The 3D input voxels V are processed by a series of 3D convolutions and a reshaping unit that transforms the 3D features into 2D by reshaping its last two dimensions (*e.g.* $h \times w \times d \times c$ becomes $h \times w \times (d * c)$), followed by 1×1 convolutions (*projection unit* in [45] and *reshape* in [48]). The reshaping step can be seen as an orthographic projection in the latent space, with all the depth information being kept. The features are then processed by a series of 2D convolutional blocks, leading to the final encoding of the input voxels.

The light input L is processed by a separate branch with 2 fully connected layers, delivering latent illumination features to the network. These features are then tiled to form an image with the same dimensions as the final features from the voxel encoding branch. In this way, every spatial location in the features has information about the illumination. After concatenation, the joint features are fed to a decoder that outputs the image in the final resolution.

Model training. We train the model by minimizing the following loss:

$$L(I, T) = ||I - T||_1 + \beta \sum_i w_i ||v_i(I) - v_i(T)||_2 \tag{2}$$

where T is the ground-truth target image. The second term is a perceptual loss, with v_i is the response of the i layer of

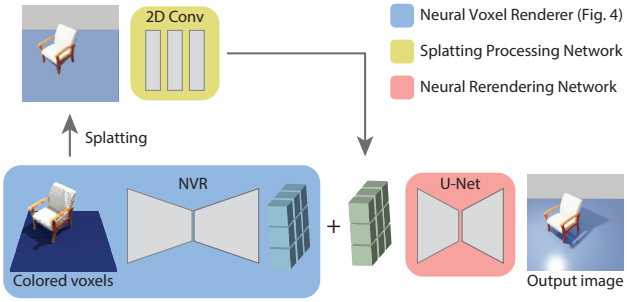


Figure 5. NVR+ network architecture.

a pretrained VGG [60] network, and w_i its weighting factor. For i, we use the *conv1, conv2* layers and set their weights w_i to 1.0 and 0.1 respectively.

The target image T is produced by a traditional, physically-based renderer (Blender Cycles [3]) and the object is represented by a 3D mesh. This results in rendering smooth surfaces in image T. In this way, the network implicitly learns to map discrete geometric representation such as voxels to a continuous and smooth rendering.

We train the network using the Adam optimizer [26] with learning rate 10^{-4} and a batch size of 10. The 2D convolutional layers are followed by batch normalization and ReLU activations (more details in supplementary material).

3.4. Adding a rerendering network (NVR+)

The network in Sec. 3.3 is able to render well the overall structure of the scene in terms of colors, reflections, shadows, etc. However, we observe that when the color pattern of the object in the input voxels forms a high frequency and irregular texture, the output is blurry and with artifacts. For this reason we propose a rerendering network (NVR+) that maintains the high quality textures while producing the correct overall scene appearance (Fig. 5).

The texture information is already encoded in the voxels' colors and we know the camera parameters of the target rendering (the user sets the scene to be rendered, see Sec. 3.1). Therefore we can synthesize an image S by splatting [72] the center of the colored voxels to an empty canvas in the target view. Note that this image will contain the artifacts mentioned in Sec. 3.2 (wrong colors, different shading, *etc.*) since the target view can be different from the view the appearance was captured from.

The NVR+ consists of three parts: the NVR network described in the previous subsection, the Splatting Processing Network that encodes the synthesized image S into a latent representation, and a Neural Rerendering Network that combines and processes the outputs of the other two networks into the final image. The Splatting Processing Network consists of a series of convolutional layers without decreasing the resolution. The output of this network is then added to the features from the NVR and the result is fed to the Neural Rerendering Network. The Neural Rerendering

Network processes the combined features with a U-Net [56] architecture and outputs the image in the final resolution. The whole NVR+ network is trained end-to-end, using the same loss as in Eq. (2).

The NVR+ is able to render high-frequency textures accurately and in detail because it combines the best of two modalities. First, the output of the NVR produces a realistic image in terms of reflections, shadows and overall color assignments but it lacks the high-frequency texture details. Second, the output of the Splatting Processing Network contain artifacts from the splatting process, but it also includes features rich in resolution and details. Finally, the Neural Rerendering Network integrates the two network outputs and produces a coherent, detailed and artifact-free final image. The NVR+ renders an image (256×256) in ≈ 0.1 sec on a single desktop GPU (Nvidia RTX).

4. Experiments

4.1. Settings and protocol

Training. For training our models we use 3D shapes from ShapeNet [11] and we render them with Blender Cycles [3], a physically-based path tracer. We focus mainly on the Chairs category, but we also provide qualitative results for the Car category. In all cases the training sets are constructed by sampling randomly 2000 3D objects from the train set as specified in the SHREC'16 challenge [58]. Each object is rendered from 20 different viewpoints by uniformly sampling (1) the elevation of the camera, (2) the rotation and translation of the object, and (3) the position of the light. The camera elevation is between 5 and 50 degrees, the object rotation is uniformly sampled from the 180 degrees hemisphere facing towards the camera, the translation of the object is sampled from an rectangular area around the scene center ($[-0.5, 0.5]$ units) and the light is sampled from a volume above the scene center ($[-1.5, 1.5]$ for the x and z axis and $[2.5, 3]$ for the y axis). The object is rendered as a mesh for accurate reproduction of its surfaces (giving the target image T in Eq. (2)). In contrast, the object is input to the network as a 100^3 voxel grid, and then it is placed inside the overall 128^3 voxel scene V to allow rotations and translations (object to world coordinates, see Fig. 2).

Testing. For testing, we want to measure the ability of our model to cope with changes in object rotation and translation, and light translation. We randomly select 40 3D objects from the test set of the SHREC'16 challenge [58]. Each object is rendered with the following settings: the camera elevation is randomly sampled between 15 and 45 degrees; a rotation angle range (starting and ending angle) is randomly sampled from the hemisphere facing the camera and the object is rotated with a step of 10 degrees; a start and end location is randomly sampled and the object is being translated between the two end points; similar sampling is applied for the light. This procedure results in 40000 images for training and 1100 images for testing.

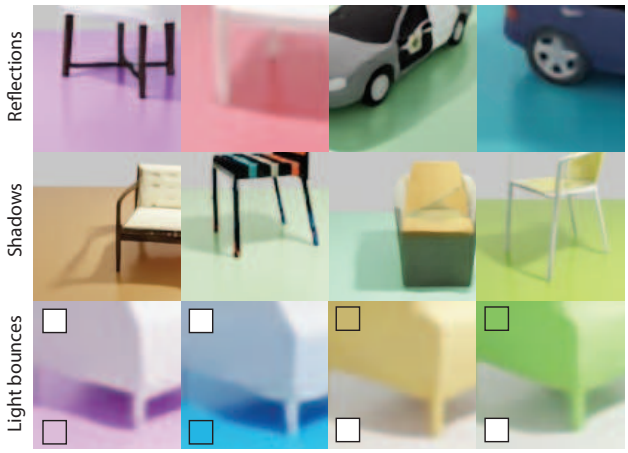


Figure 6. Global illumination effects produced by our framework.

Appearance settings. We generate three settings with varying appearance complexity. *Single color*: where both the object and the floor have a single randomly selected color (RGB values); *Default*: where the object is rendered with its default ShapeNet textures/materials but the floor color is randomly sampled; and *Textured*: where the object has a randomly selected texture from the Describable Textures Dataset[13] (we separate the textures in train and test splits) and the floor has a single randomly selected color. We train a model for every setting and every category.

4.2. Global illumination effects

Our scenes consists of objects that interact with each other (shadows, reflections, *etc*.) and our training data was generated with a physically-based renderer. These elements of realism exist in the training dataset, and here we analyze if our network is able to reproduce these effects.

In Fig. 6 we illustrate how our framework renders global illumination effects. In the first row we show the reflections on the specular ground for different objects. The overall structure is represented properly and it is following the orientation of the object, even with thin structures. In the second row we show how our network is rendering shadows; again, thin structures produce thin shadows and concave objects allow the light to pass. Finally, in the last row we show the effect of multiple light bounces: the color of the object (first two columns) and the ground (last two columns) is affected by the color of the other object. The original color of the object is shown in the small rectangle inset.

4.3. Neural rendering analysis

In this subsection we analyze the design choices from Section 3 and evaluate their effects on the final renderings. We additionally compare with alternative techniques for neural rendering and show that our framework performs better both quantitatively and qualitatively.

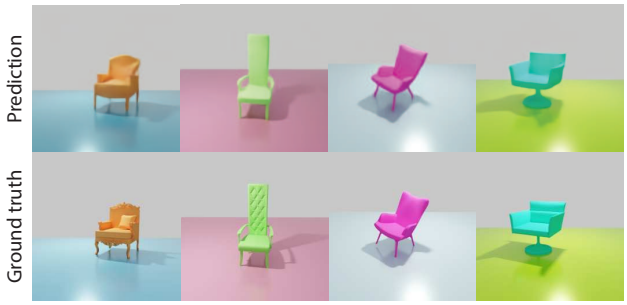


Figure 7. Results of NVR for the *single color* setting. The voxel colors are properly mapped to pixels in the output images.

Manual coloring. In this scenario, we use the *Single color* appearence setting as training data. With this experiment, we want to illustrate the capability of our method (NVR) to map the voxel colors to rendered pixels. Fig. 7 shows that the networks learns to render accurately the object, with correct colors and shading.

Colors from image. In this scenario, the voxels take their color from an input image (appearance source image $\mathcal{A}$, see Sec. 3.2). This is a more challenging setting, as some parts of the object are hidden and the colors contain view-dependent appearance information. Hence, it forms a good test case for evaluating the ability to render more complex appearance (different parts, textures, *etc*.) while capturing light interactions among scene elements. Fig. 8 shows the output of our NVR and NVR+ models in the *default* appearance setting for the Chair category (5th and and 6th column). The NVR model properly assigns the colors to the individual parts, but fine-grained details on textured areas are washed out. The NVR+ model renders detailed textures while also accurately producing object shading, ground reflections and shadows. Fig. 9 (left) shows the output of NVR+ for the *default* appearance setting of the Car category (using a car-specific model, but with different Car instances for training and testing, Sec. 4.1). In the first row we show the voxel input[1] together with the appearance source image $\mathcal{A}$ from where the colors were taken, in second row our prediction and in third row the ground truth. Again, our method can reproduce accurately the details of the cars; also, by taking advantage of the symmetry, we are able to faithfully reconstruct the hidden side. Finally, we visualize the output of NVR+ on the *textured* appearance setting (Fig. 9 right). As the images show, this model can handle high frequency and irregular textures (as well as synthesizing specular reflections and shadows as in Fig. 8). Moreover, this confirms that our model does not memorize the training data but rather learns to map the input color pattern information into an accurate rendered image.

Comparison to alternative methods. We use the Chair category (*default* appearance setting) and for every 3D object, one of the rendered images acts as the appearance source image $\mathcal{A}$. We compare against a set of alternatives based on recent works on neural rendering. Since there is no other method that offers control over geometric, appearance and light edits, we modify them to be comparable.

Projective texturing: here we use the alignment between the mesh and the input image to texture the mesh (i.e. estimate its UV map). In Fig. 10 we illustrate typical projective texturing artifacts: a) the hidden areas copy the appearance of the visible parts, even when taking symmetry into account (red arrows), and b) light is "baked" into the object appearance, making difficult to render with new lighting.

Single-to-mutliview: this method is based on [64] and consists of a U-net which inputs the appearance source image $\mathcal{A}$, the relative object rotation/translation and light position. This method does not use any 3D information.

Image encoding: here we have a setup similar to [45] but instead of painting the voxels directly, we pass $\mathcal{A}$ through a network to get a latent code that is then appended to the input voxels (similar to how [45] rendered textured faces).

[1] rendered with MagicaVoxel [5] in 4 seconds each using a ray-tracer.

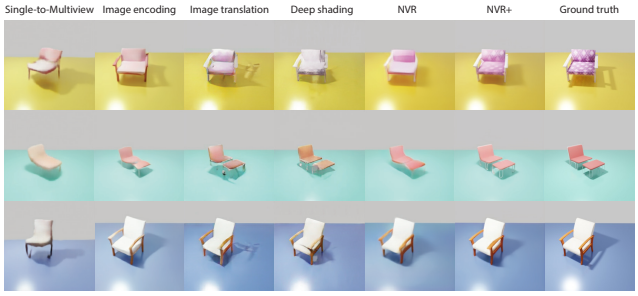


Figure 8. Visual comparison between our models and different methods (see text for details).

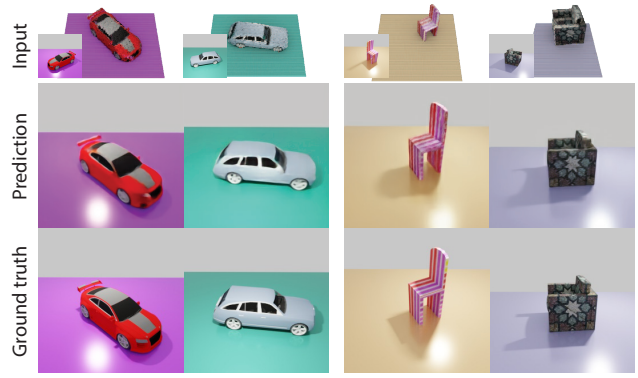


Figure 9. Neural rendering of cars and textured objects with NVR+.

Figure 10. Artifacts of projective texturing.

Method	MSE	DSSIM	Perceptual
Single-to-multiview	34.8	0.040	40.17
Image encoding	24.3	0.030	24.41
Ours (NVR)	22.8	0.028	21.25
Image translation	21.6	0.026	18.12
Deep shading	21.8	0.024	16.54
Ours (NVR+)	**14.3**	**0.012**	**8.21**

Table 1. Evaluation of different methods for neural rendering.

The rest of the network has similar architecture to NVR so the light can be given as an input.

Image translation: this method is based on [23]; the input is the rerendered image S (generated by splatting the voxels onto the image plane) together with the light position as additional channel. The desired output is the target image T.

Deep shading: here the setup is similar to [44]. We use projective texturing to estimate the UV map of the mesh; then we place the mesh in the desired position and we render the diffuse color and depth buffer. We use these buffers together with the light position as inputs to a Pix2Pix [23] network and we optimize for the target image T.

Table 1 compares the performance of the different methods on mean squared error (MSE), structural dissimilarity (DSSIM) and the perceptual loss in Eq. (2) (using the same layers i and weights w_i). NVR+ performs better than the alternatives in all metrics by a large margin. This illustrates the ability of the rerendering module to reproduce fine details, especially for textured 3D objects. Fig. 8 provides a qualitative comparison of the different methods. As can be seen, *Single-to-mutliview* results are blurry, while *Image encoding* captures the overall color but fail to assign it correctly to the individual parts. The *Image translation* model produces typical GAN artifacts and cannot estimate the shadows properly. The *Deep shading* model also faces similar limitations, despite using a mesh representation instead of voxels. Our NVR model captures the color and structure of the scene, but smooths out the fine texture details. Finally, our NVR+ model accurately renders both the geometry and the texture of the object, while also realistically synthesizing shadows, specular reflections and highlights in the output image.

Voxel resolution. The object has an initial voxel resolution of 100^3 and is then placed in a scene $V \in 128^{3\times 4}$. Here, we investigate the rendering quality when the initial voxel resolution varies. In Table 2 and Fig. 11 we show the performance of smaller resolutions for the NVR+ model (which are then rescaled with nearest neighbor interpolation). The performance decreases gracefully and even at a very low resolution (25^3) our method produces plausible outputs.

4.4. Editing analysis

Illumination edits. When the light source changes position, the scene appearance should change accordingly. In Fig. 12, we illustrate this effect: as the light sources moves, the brightness of different parts of the object changes and shadows/light reflections in the ground move accordingly.

	100^3	75^3	50^3	25^3
MSE	14.3	14.4	15.3	19.3
DSSIM	0.012	0.014	0.016	0.024
Perceptual	8.21	9.47	10.9	18.68

Table 2. Effect of voxel resolution on performance.

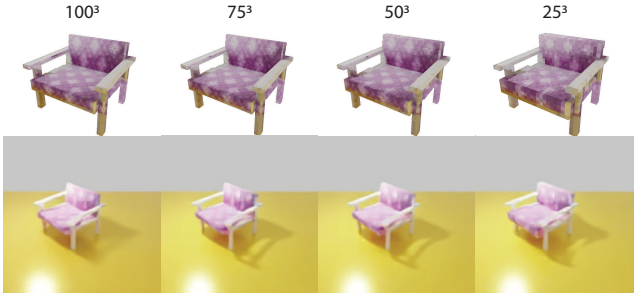


Figure 11. Rendering an object with different voxel resolutions.

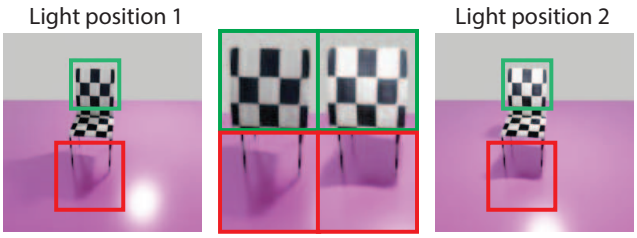


Figure 12. Illumination effects by changing the light position. Our framework changes properly the overall shading (e.g. the back of the chair is brighter in position 2) and the shadows.

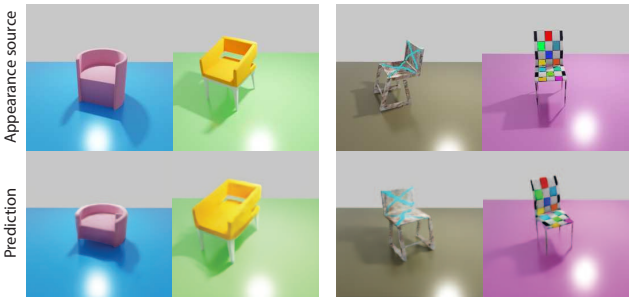


Figure 13. Applying geometric (left) and appearance (right) modifications.

Figure 14. Neural rendering with natural illumination as an input.

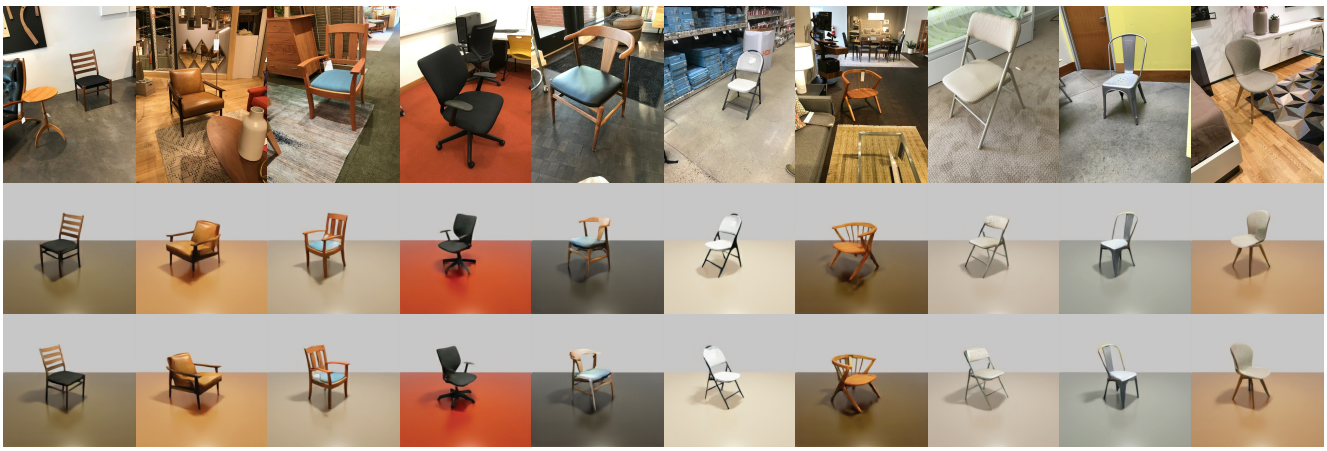


Figure 15. Rendering real objects. In the first row there are the appearance source images and in second and third row renderings of the objects using the NVR+ network.

Figure 16. Rendering different categories. The categories of these real objects were not part of the training dataset (trained only on the chair category).

Object geometry edits. Apart from global object rotations and translations, we can also deform the object. In Fig. 13 left we visualize the effect of scaling across an axis, resulting in elongated or squeezed versions of the object.

Object appearance edits. Detailed modifications on the appearance source image can propagate through our neural renderer. In this example, we manually paint patterns and letters on the appearance source image, so during the coloring step (Sec. 3.2) the edits pass on to the voxels. Fig. 13 right shows how our method can synthesizes images with the object in new viewpoints and the light in new positions while preserving these fine-grained edits that were made to the appearance source image.

Increasing realism. In this experiment, we investigate the use of more realistic ways to illuminate the scene. We modify the NVR+ network so instead of the xyz light coordinates, it takes as an input an 32×32 environment map. The environment map is processed by a series of convolutional layers to extract a latent code, which is then supplied to the NVR+ network. We additionally consider a textured circular ground and add specularity to the *default* dataset objects. We use 80 environment maps for training and 20 for testing, taken from [4]. Results are shown in Fig. 14, with the appearance source image shown in an inset.

Appearance capture from real images. So far we have experimented with synthetic images with varying appearance complexity. However, our framework can capture object appearance from any input image. In this experiment, we use the Pix3D dataset [63] which contains aligned pairs of images with 3D objects. We use the real image as the appearance source and we map it to to the voxels of the provided 3D object as before. Note that unlike the previous experiments with ShapeNet objects, Pix3D also includes scanned objects with imperfect geometry. In Fig. 15 we present our renderings when the voxels and the appearance come from real images. Our framework is able to faithfully render these objects despite not being trained on real objects and despite significantly different geometric, illumination and material conditions than the training set.

Testing on other categories. Our framework extends to other categories that were not included in the training dataset. In this experiment we take the NVR+ network trained on the default Chair category and we apply it to real images from other categories. In Fig. 16 we illustrate the rendering for sofas, tables and other miscellaneous objects from Ikea [33] and Pix3D [63] datasets.

5. Conclusion

We presented Neural Voxel Renderer, a framework that synthesizes realistic images given object voxels as input, and provides editing functionalities to the output. Our framework can reproduce the detailed appearance of the input due to a rerendering module that handles high-frequency and complex textures. We show a wide range of rendering scenarios, where we modify the input scene with respect to illumination, object geometry and appearance. Moreover, we demonstrate the appearance capture and rendering of real objects from several categories. We hence believe that our neural renderer is a useful tool that advances the state-of-the-art and can spawn further research.

References

[1] Adobe substance. https://www.substance3d.com/. 2

[2] Autodesk 3ds max. https://www.autodesk.com/products/3ds-max/. 1

[3] Blender - a 3d modelling and rendering package. http://www.blender.org. 1, 4, 5

[4] Hdri-haven. https://hdrihaven.com/. 8

[5] Magicavoxel. https://ephtracy.github.io/. 3, 6

[6] Unity photogrammetry workflow. https://unity.com/solutions/photogrammetry. 2

[7] Mathieu Aubry, Daniel Maturana, Alexei Efros, Bryan Russell, and Josef Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *CVPR*, 2014. 3

[8] Guha Balakrishnan, Amy Zhao, Adrian V. Dalca, Frédo Durand, and John V. Guttag. Synthesizing images of humans in unseen poses. In *CVPR*, 2018. 2

[9] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019. 1

[10] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. Everybody dance now. In *ICCV*, 2019. 2

[11] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 5

[12] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaako Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *NeurIPS*, 2019. 2

[13] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *CVPR*, 2014. 5

[14] Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. Single-image svbrdf capture with a rendering-aware deep network. *ACM Trans. Graph.*, 37(4), 2018. 3

[15] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *CVPR*, 2015. 2

[16] Ersin Yumer Duygu Ceylan Alexander C. Berg Eunbyung Park, Jimei Yang. Transformation-grounded image generation network for novel 3d view synthesis. In *CVPR*, 2017. 2

[17] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *CVPR*, Jun 2016. 2

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*. 2014. 1

[19] Yoshitaka Ushiku Hiroharu Kato and Tatsuya Harada. Neural 3d mesh renderer. In *CVPR*, 2018. 2, 3

[20] Hui Huang, Ke Xie, Lin Ma, Dani Lischinski, Minglun Gong, Xin Tong, and Daniel Cohen-or. Appearance modeling via proxy-to-image alignment. *ACM Trans. Graph.*, 37(1):10:1–10:15, 2018. 3

[21] Qixing Huang, Hai Wang, and Vladlen Koltun. Single-view reconstruction via joint analysis of image and shape collections. *ACM Trans. Graph.*, 34, 08 2015. 3

[22] Moos Hueting, Pradyumna Reddy, Ersin Yumer, Vladimir G. Kim, Nathan Carr, and Niloy J. Mitra. Seethrough: Finding objects in heavily occluded indoor scene images. In *3DV*, 2018. 3

[23] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. 1, 2, 7

[24] James T. Kajiya. The rendering equation. In *SIGGRAPH*, 1986. 1

[25] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019. 1

[26] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 4

[27] Tejas D Kulkarni, William F. Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *NeurIPS*. 2015. 2

[28] Marc Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3), 1988. 3

[29] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph.*, 37(6), 2018. 2

[30] Xiao Li, Yue Dong, Pieter Peers, and Xin Tong. Modeling surface appearance from a single photograph using self-augmented convolutional neural networks. *ACM Trans. Graph.*, 36(4), 2017. 3

[31] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. In *ECCV*, 2018. 2

[32] Zhengqin Li, Kalyan Sunkavalli, and Manmohan Krishna Chandraker. Materials for masses: Svbrdf acquisition with a single mobile phone image. In *ECCV*, 2018. 3

[33] Joseph J. Lim, Hamed Pirsiavash, and Antonio Torralba. Parsing IKEA Objects: Fine Pose Estimation. In *ICCV*, 2013. 8

[34] Guilin Liu, Duygu Ceylan, Ersin Yumer, Jimei Yang, and Jyh-Ming Lien. Material editing using a physically based rendering network. *ICCV*, 2017. 3

[35] L. Liu, W. Xu, M. Zollhoefer, H. Kim, F. Bernard, M. Habermann, W. Wang, and C. Theobalt. Neural Animation and Reenactment of Human Actor Videos. *ACM Trans. Graph.*, 2019. 2

[36] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *ICCV*, 2019. 2, 3

[37] Stephen Lombardi, Jason Saragih, Tomas Simon, and Yaser Sheikh. Deep appearance models for face rendering. *ACM Trans. Graph.*, 37(4), 2018. 2

[38] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural vol-

umes: Learning dynamic renderable volumes from images. *ACM Trans. Graph.*, 38(4), 2019. 2

[39] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. In *CVPR*, 2017. 2

[40] Liqian Ma, Xu Jia, Qianru Sun, Bernt Schiele, Tinne Tuytelaars, and Luc Van Gool. Pose guided person image generation. In *NeurIPS*, 2017. 2

[41] Ricardo Martin-Brualla, Rohit Pandey, Shuoran Yang, Pavel Pidlypenskyi, Jonathan Taylor, Julien Valentin, Sameh Khamis, Philip Davidson, Anastasia Tkach, Peter Lincoln, Adarsh Kowdle, Christoph Rhemann, Dan B Goldman, Cem Keskin, Steve Seitz, Shahram Izadi, and Sean Fanello. Lookingood: Enhancing performance capture with real-time neural re-rendering. *ACM Trans. Graph.*, 37(6), 2018. 1, 2

[42] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*, 2019. 3

[43] Moustafa Meshry, Dan B Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. Neural rerendering in the wild. In *CVPR*, 2019. 1, 2

[44] Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, Hans-Peter Seidel, and Tobias Ritschel. Deep shading: Convolutional neural networks for screen-space shading. 36(4), 2017. 1, 2, 3, 7

[45] Thu Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yong-Liang Yang. Rendernet: A deep convolutional network for differentiable rendering from 3d shapes. In *NeurIPS*, 2018. 1, 2, 4, 6

[46] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. In *ICCV*, 2019. 2

[47] Michael Oechsle, Lars M. Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *ICCV*, 2019. 2, 3

[48] Kyle Olszewski, Sergey Tulyakov, Oliver Woodford, Hao Li, and Linjie Luo. Transformable bottleneck networks. 2019. 2, 4

[49] Jeong Joon Park, Peter Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019. 3

[50] Keunhong Park, Konstantinos Rematas, Ali Farhadi, and Steven M. Seitz. Photoshape: Photorealistic materials for large-scale shape collections. *ACM Trans. Graph.*, 37(6), Nov. 2018. 3

[51] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019. 2

[52] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6), 1975. 2

[53] Francesco Pittaluga, Sanjeev J Koppal, Sing Bing Kang, and Sudipta N Sinha. Revealing scenes by inverting structure from motion reconstructions. In *CVPR*, 2019. 2

[54] Konstantinos Rematas, Chuong Nguyen, Tobias Ritschel, Mario Fritz, and Tinne Tuytelaars. Novel views of objects from a single image. *TPAMI*, 2017. 3

[55] Konstantinos Rematas, Tobias Ritschel, Mario Fritz, and Tinne Tuytelaars. Image-based synthesis and re-synthesis of viewpoints guided by 3d models. In *CVPR*, 2014. 2

[56] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 5

[57] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015. 1

[58] M. Savva, F. Yu, Hao Su, M. Aono, B. Chen, D. Cohen-Or, W. Deng, Hang Su, S. Bai, X. Bai, N. Fish, J. Han, E. Kalogerakis, E. G. Learned-Miller, Y. Li, M. Liao, S. Maji, A. Tatsuma, Y. Wang, N. Zhang, and Z. Zhou. Large-scale 3d shape retrieval from shapenet core55. In *Eurographics Workshop on 3D Object Retrieval*, 2016. 5

[59] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR*, 2006. 2

[60] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 4

[61] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. Deepvoxels: Learning persistent 3d feature embeddings. In *CVPR*, 2019. 2

[62] Shao-Hua Sun, Minyoung Huh, Yuan-Hong Liao, Ning Zhang, and Joseph J Lim. Multi-view to novel view: Synthesizing novel views with self-learned confidence. In *ECCV*, 2018. 2

[63] Xingyuan Sun, Jiajun Wu, Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Tianfan Xue, Joshua B Tenenbaum, and William T Freeman. Pix3d: Dataset and methods for single-image 3d shape modeling. In *CVPR*, 2018. 8

[64] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Multi-view 3d models from single images with a convolutional network. In *ECCV*, 2016. 2, 6

[65] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Trans. Graph.*, 2019. 2

[66] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *CVPR*, 2017. 2

[67] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *CVPR*, 2018. 2

[68] Tuanfeng Y. Wang, Hao Su, Qixing Huang, Jingwei Huang, Leonidas Guibas, and Niloy J. Mitra. Unsupervised texture transfer from images to model collections. *ACM Trans. Graph.*, 35(6), 2016. 3

[69] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Interpretable transformations with encoder-decoder networks. In *ICCV*, 2017. 2

[70] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Trans. Graph.*, 38(6), 2019. 2

[71] Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. View synthesis by appearance flow. In *ECCV*, 2016. 2

[72] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus H. Gross. Surface splatting. In *SIGGRAPH*, 2001. 4