Neural 3D Mesh Renderer

Hiroharu Kato[1], Yoshitaka Ushiku[1], and Tatsuya Harada[1,2]
[1]The University of Tokyo, [2]RIKEN
{kato,ushiku,harada}@mi.t.u-tokyo.ac.jp

Abstract

For modeling the 3D world behind 2D images, which 3D representation is most appropriate? A polygon mesh is a promising candidate for its compactness and geometric properties. However, it is not straightforward to model a polygon mesh from 2D images using neural networks because the conversion from a mesh to an image, or rendering, involves a discrete operation called rasterization, which prevents back-propagation. Therefore, in this work, we propose an approximate gradient for rasterization that enables the integration of rendering into neural networks. Using this renderer, we perform single-image 3D mesh reconstruction with silhouette image supervision and our system outperforms the existing voxel-based approach. Additionally, we perform gradient-based 3D mesh editing operations, such as 2D-to-3D style transfer and 3D DeepDream, with 2D supervision for the first time. These applications demonstrate the potential of the integration of a mesh renderer into neural networks and the effectiveness of our proposed renderer.

1. Introduction

Understanding the 3D world from 2D images is one of the fundamental problems in computer vision. Humans model the 3D world in their brains using images on their retinas, and live their daily existence using the constructed model. The machines, too, can act more intelligently by explicitly modeling the 3D world behind 2D images.

The process of generating an image from the 3D world is called *rendering*. Because this lies on the border between the 3D world and 2D images, it is crucially important in computer vision.

In recent years, convolutional neural networks (CNNs) have achieved considerable success in 2D image understanding [7, 13]. Therefore, incorporating rendering into neural networks has a high potential for 3D understanding.

What type of 3D representation is most appropriate for modeling the 3D world? Commonly used 3D formats are voxels, point clouds and polygon meshes. Voxels, which

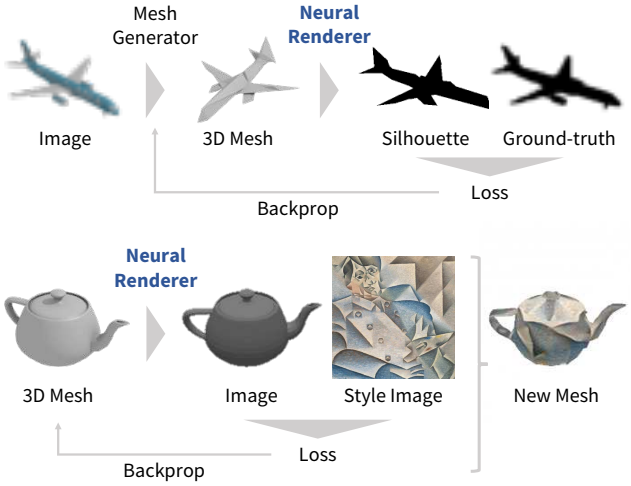

Figure 1. Pipelines for single-image 3D mesh reconstruction (upper) and 2D-to-3D style transfer (lower).

are 3D extensions of pixels, are the most widely used format in machine learning because they can be processed by CNNs [2, 17, 20, 24, 30, 31, 34, 35, 36]. However, it is difficult to process high resolution voxels because they are regularly sampled from 3D space and their memory efficiency is poor. The scalability of point clouds, which are sets of 3D points, is relatively high because point clouds are based on irregular sampling. However, textures and lighting are difficult to apply because point clouds do not have surfaces. Polygon meshes, which consist of sets of vertices and surfaces, are promising because they are scalable and have surfaces. Therefore, in this work, we use the polygon mesh as our 3D format.

One advantage of polygon meshes over other representations in 3D understanding is its compactness. For example, to represent a large triangle, a polygon mesh only requires three vertices and one face, whereas voxels and point clouds require many sampling points over the face. Because polygon meshes represent 3D shapes with a small number of parameters, the model size and dataset size for 3D understanding can be made smaller.

Another advantage is its suitability for geometric trans-

formations. The rotation, translation, and scaling of objects are represented by simple operations on the vertices. This property also facilitates to train 3D understanding models.

Can we train a system including rendering as a neural network? This is a challenging problem. Rendering consists of projecting the vertices of a mesh onto the screen coordinate system and generating an image through regular grid sampling [16]. Although the former is a differentiable operation, the latter, referred to as *rasterization*, is difficult to integrate because back-propagation is prevented by the discrete operation.

Therefore, to enable back-propagation with rendering, we propose an approximate gradient for rendering peculiar to neural networks, which facilitates end-to-end training of a system including rendering. Our proposed renderer can flow gradients into texture, lighting, and cameras as well as object shapes. Therefore, it is applicable to a wide range of problems. We name our renderer *Neural Renderer*.

In the generative approach in computer vision and machine learning, problems are solved by modeling and inverting the process of data generation. Images are generated via rendering from the 3D world, and a polygon mesh is an efficient, rich and intuitive 3D representation. Therefore, "backward pass" of mesh renderers is extremely important.

In this work, we propose the two applications illustrated in Figure 1. The first is single-image 3D mesh reconstruction with silhouette image supervision. Although 3D reconstruction is one of the main problems in computer vision, there are few studies to reconstruct meshes from single images despite the potential capacity of this approach. The other application is gradient-based 3D mesh editing with 2D supervision. This includes a 3D version of style transfer [6] and DeepDream [18]. This task cannot be realized without a differentiable mesh renderer because voxels or point clouds have no smooth surfaces.

The major contributions can be summarized as follows.

- We propose an approximate gradient for rendering of a mesh, which enables the integration of rendering into neural networks.
- We perform 3D mesh reconstruction from single images without 3D supervision and demonstrate our system's advantages over the voxel-based approach.
- We perform gradient-based 3D mesh editing operations, such as 2D-to-3D style transfer and 3D DeepDream, with 2D supervision for the first time.
- We will release the code for Neural Renderer.

2. Related work

In this section, we briefly describe how 3D representations have been integrated into neural networks. We also summarize works related to our two applications.

2.1. 3D representations in neural networks

3D representations are categorized into rasterized and geometric forms. Rasterized forms include voxels and multi-view RGB(D) images. Geometric forms include point clouds, polygon meshes, and sets of primitives.

Rasterized forms are widely used because they can be processed by CNNs. Voxels, which are 3D extensions of pixels, are used for classification [17, 20, 24, 34, 35], 3D reconstruction and generation [2, 30, 31, 34, 36]. Because the memory efficiency of voxels is poor, some recent works have incorporated more efficient representations [24, 30, 32]. Multi-view RGB(D) images, which represent a 3D scene through a set of images, are used for recognition [20, 27] and view synthesis [29].

Geometric forms require some modifications to be integrated into neural networks. For example, systems that handle point clouds must be invariant to the order of points. Point clouds have been used for both recognition [12, 19, 21] and reconstruction [5]. Primitive-based representations, which represent 3D objects using a set of primitives, such as cuboids, have also been investigated [14, 39].

A Polygon mesh represents a 3D object as a set of vertices and surfaces. Because it is memory efficient, suitable for geometric transformations, and has surfaces, it is the de facto standard form in computer graphics (CG) and computer-aided design (CAD). However, because the data structure of a polygon mesh is a complicated graph, it is difficult to integrate into neural networks. Although recognition and segmentation have been investigated [10, 38], generative tasks are much more difficult. Rezende *et al*. [23] incorporated the OpenGL renderer into a neural network for 3D mesh reconstruction. Gradients of the black-box renderer were estimated using REINFORCE [33]. In contrast, the gradients in our renderer are geometry-grounded and presumably more accurate. OpenDR [15] is a differentiable renderer. Unlike this general-purpose renderer, our proposed gradients are designed for neural networks.

2.2. Single-image 3D reconstruction

The estimation of 3D structures from images is a traditional problem in computer vision. Following the recent progress in machine learning algorithms, 3D reconstruction from a single image has become an active research topic.

Most methods learn a 2D-to-3D mapping function using ground truth 3D models. While some works reconstruct 3D structures via depth prediction [4, 25], others directly predict 3D shapes [2, 5, 30, 31, 34].

Single-image 3D reconstruction can be realized without 3D supervision. Perspective transformer nets (PTN) [36] learn 3D structures using silhouette images from multiple viewpoints. Our 3D reconstruction method is also based on silhouette images. However, we use polygon meshes whereas they used voxels.

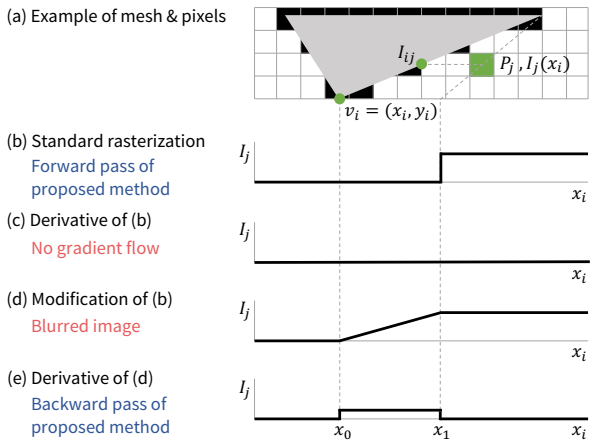

Figure 2. Illustration of our method. $\boldsymbol{v}_i = \{x_i, y_i\}$ is one vertex of the face. I_j is the color of pixel P_j. The current position of x_i is x_0. x_1 is the location of x_i where an edge of the face collides with the center of P_j when x_i moves to the right. I_j becomes I_{ij} when $x_i = x_1$.

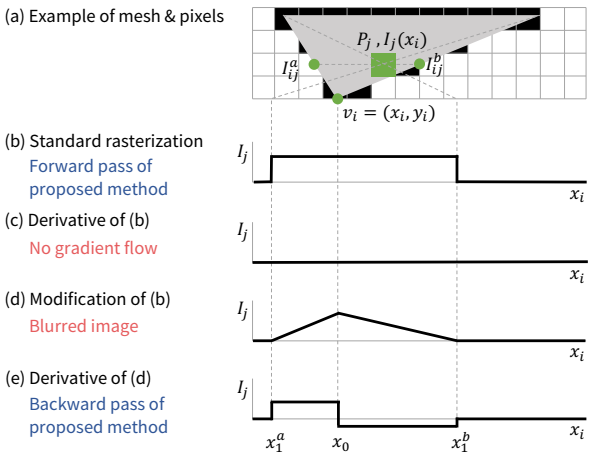

Figure 3. Illustration of our method in the case where P_j is inside the face. I_j changes when x_i moves to the right or left.

2.3. Image editing via gradient descent

Using a differentiable feature extractor and loss function, an image that minimizes the loss can be generated via back-propagation and gradient descent. DeepDream [18] is an early example of such a system. An initial image is repeatedly updated so that the magnitude of its image feature becomes larger. Through this procedure, objects such as dogs and cars gradually appear in the image.

Image style transfer [6] is likely the most familiar and practical example. Given a *content image* and *style image*, an image with the specified content and style is generated.

Our renderer provides gradients of an image with respect to the vertices and textures of a mesh. Therefore, DeepDream and style transfer of a mesh can be realized by using loss functions on 2D images.

3. Approximate gradient for rendering

In this section, we describe Neural Renderer, which is a 3D mesh renderer with gradient flow.

3.1. Rendering pipeline and its derivative

A 3D mesh consists of a set of vertices $\{\boldsymbol{v}^o_1, \boldsymbol{v}^o_2, .., \boldsymbol{v}^o_{N_v}\}$ and faces $\{\boldsymbol{f}_1, \boldsymbol{f}_2, .., \boldsymbol{f}_{N_f}\}$, where the object has N_v vertices and N_f faces. $\boldsymbol{v}^o_i \in \mathbb{R}^3$ represents the position of the i-th vertex in the 3D object space and $\boldsymbol{f}_j \in \mathbb{N}^3$ represents the indices of the three vertices corresponding to the j-th triangle face. To render this object, vertices $\{\boldsymbol{v}^o_i\}$ in the object space are transformed into vertices $\{\boldsymbol{v}^s_i\}$, $\boldsymbol{v}^s_i \in \mathbb{R}^2$ in the screen space. This transformation is represented by a combination of differentiable transformations [16].

An image is generated from $\{\boldsymbol{v}^s_i\}$ and $\{\boldsymbol{f}_j\}$ via sampling. This process is called rasterization. Figure 2 (a) illustrates rasterization in the case of single triangle. In rasterization, each pixel is painted in the color of the triangle face overlapping it. Since this is a discrete operation, the color of a pixel cannot be differentiated by the position of faces. This causes a problem in back-propagation.

3.2. Rasterization of a single face

For ease of explanation, we describe our method using the x-coordinate x_i of a single vertex $\boldsymbol{v}_i = \boldsymbol{v}^s_i$ in the screen space and a single gray-scale pixel P_j. We consider the color of P_j to be a function $I_j(x_i)$ on x_i and freeze all variables other than x_i.

First, we assume that P_j is outside the face, as shown in Figure 2 (a). The color of P_j is $I(x_0)$ when x_i is at the current position x_0. If x_i moves to the right and reaches the point x_1, where an edge of the face collides with the center of P_j, $I_j(x_i)$ suddenly turns to the color of hitting point I_{ij}. Let δ^x_i be the distance traveled by x_i, let $\delta^x_i = x_1 - x_0$, and let δ^I_j represent the change in the color $\delta^I_j = I(x_1) - I(x_0)$. The partial derivative $\frac{\partial I_j(x_i)}{\partial x_i}$ is zero almost everywhere, as illustrated in Figure 2 (b–c).

Because the gradient is zero, the information that $I_j(x_i)$ can be changed by δ^I_j if x_i moves by δ^x_i to the right is not transmitted to x_i. This is because $I_j(x_i)$ suddenly changes. Therefore, we replace the sudden change with a gradual change between x_0 and x_1 using linear interpolation. Then, $\frac{\partial I_j}{\partial x_i}$ becomes $\frac{\delta^I_j}{\delta^x_i}$ between x_0 and x_1, as shown in Figure 2 (d–e).

The derivative of $I_j(x_i)$ is different on the right and left sides of x_0. How should one define a derivative at $x_i = x_0$? We propose switching the values using the error signal δ^P_j back-propagated to P_j. The sign of δ^P_j indicates whether P_j should be brighter or darker. To minimize the loss, if

$\delta_j^P > 0$, then P_j must be darker. On the other hand, the sign of δ_j^I indicates whether P_j can be brighter or darker. If $\delta_j^I > 0$, P_j becomes brighter by pulling in x_i, but P_j cannot become darker by moving x_i. Therefore, a gradient should not flow if $\delta_j^P > 0$ and $\delta_j^I > 0$. From this viewpoint, we define $\frac{\partial I_j(x_i)}{\partial x_i}|_{x_i=x_0}$ as follows.

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0} = \begin{cases} \frac{\delta_j^I}{\delta_i^x}; & \delta_j^P \delta_j^I < 0. \\ 0; & \delta_j^P \delta_j^I \geq 0. \end{cases} \tag{1}$$

Sometimes, the face does not overlap P_j regardless of where x_i moves. This means that x_1 does not exist. In this case, we define $\frac{\partial I_j(x_i)}{\partial x_i}|_{x_i=x_0} = 0$.

We use Figure 2 (b) for the forward pass because if we use Figure 2 (d), the color of a face leaks outside of the face. Therefore, our rasterizer produces the same images as the standard rasterizer, but it has non-zero gradients.

The derivative with respect to y_i can be obtained by swapping the x-axis and y-axis in the above discussion.

Next, we consider a case where P_j is inside the face, as shown in Figure 3 (a). In this case, $I(x_i)$ changes when x_i moves to the right or left. Standard rasterization, its derivative, an interpolated function, and its derivative are shown in Figure 3 (b–e). We first compute the derivatives on the left and right sides of x_0 and let their sum be the gradient at x_0. Specifically, using the notation in Figure 3, $\delta_j^{I^a} = I(x_1^a) - I(x_0)$, $\delta_j^{I^b} = I(x_1^b) - I(x_0)$, $\delta_x^a = x_1^a - x_0$ and $\delta_x^b = x_1^b - x_0$, we define the loss as follows.

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0} = \left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0}^a + \left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0}^b. \tag{2}$$

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0}^a = \begin{cases} \frac{\delta_j^{I^a}}{\delta_x^a}; & \delta_j^P \delta_j^{I^a} < 0. \\ 0; & \delta_j^P \delta_j^{I^a} \geq 0. \end{cases} \tag{3}$$

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0}^b = \begin{cases} \frac{\delta_j^{I^b}}{\delta_x^b}; & \delta_j^P \delta_j^{I^b} < 0. \\ 0; & \delta_j^P \delta_j^{I^a} \geq 0. \end{cases} \tag{4}$$

3.3. Rasterization of multiple faces

If there are multiple faces, our rasterizer draws only the frontmost face at each pixel, which is the same as the standard method [16]. During the backward pass, we first check whether or not the cross points I_{ij}, I_{ij}^a, and I_{ij}^b are drawn, and do not flow gradients if they are occluded by surfaces not including $\boldsymbol{v}_i$.

3.4. Texture

Textures can be mapped onto faces. In our implementation, each face has its own texture image of size $s_t \times s_t \times s_t$. We determine the coordinates in the texture space corresponding to a position $\boldsymbol{p}$ on a triangle $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3\}$ using the centroid coordinate system. In other words, if $\boldsymbol{p}$ is expressed as $\boldsymbol{p} = w_1\boldsymbol{v}_1 + w_2\boldsymbol{v}_2 + w_3\boldsymbol{v}_3$, let (w_1, w_2, w_3) be the corresponding coordinates in the texture space. Bilinear interpolation is used for sampling from a texture image.

3.5. Lighting

Lighting can be applied directly to a mesh, unlike voxels and point clouds. In this work, we use a simple ambient light and directional light without shading. Let l^a and l^d be the intensities of the ambient light and directional light, respectively, $\boldsymbol{n}^d$ be a unit vector indicating the direction of the directional light, and $\boldsymbol{n}_j$ be the normal vector of a surface. We then define the modified color of a pixel I_j^l on the surface as $I_j^l = \left(l^a + \left(\boldsymbol{n}^d \cdot \boldsymbol{n}_j\right) l^d\right) I_j$.

In this formulation, gradients also flow into the intensities l^a and l^d, as well as the direction $\boldsymbol{n}^d$ of the directional light. Therefore, light sources can also be included as an optimization target.

4. Applications of Neural Renderer

We apply our proposed renderer to (a) single-image 3D reconstruction with silhouette image supervision and (b) gradient-based 3D mesh editing, including a 3D version of style transfer [6] and DeepDream [18]. An image of a mesh m rendered from a viewpoint ϕ_i is denoted $R(m, \phi_i)$.

4.1. Single image 3D reconstruction

Yan *et al.* [36] demonstrated that single-image 3D reconstruction can be realized without 3D training data. In their setting, a 3D generation function $G(x)$ on an image x was trained such that silhouettes of a predicted 3D shape $\{\hat{s}_i = R(G(x), \phi_i)\}$ match the ground truth silhouettes $\{s_i\}$, assuming that the viewpoints $\{\phi_i\}$ are known. This pipeline is illustrated in Figure 1. While Yan *et al.* [36] generated voxels, we generate a mesh.

Although voxels can be generated by extending existing image generators [8, 22] to the 3D space, mesh generation is not so straightforward. In this work, instead of generating a mesh from scratch, we deform a predefined mesh to generate a new mesh. Specifically, we use an isotropic sphere with 642 vertices and move each vertex $\boldsymbol{v}_i$ as $\boldsymbol{v}_i + \boldsymbol{b}_i + \boldsymbol{c}$ using a local bias vector $\boldsymbol{b}_i$ and global bias vector $\boldsymbol{c}$. Additionally, we restrict the movable range of each vertex within the same quadrant on the original sphere. The faces $\{\boldsymbol{f}_i\}$ are unchanged. Therefore, the intermediate outputs of $G(x)$ are $\boldsymbol{b} \in \mathbb{R}^{642\times 3}$ and $\boldsymbol{c} \in \mathbb{R}^{1\times 3}$. The mesh we use is specified by 642×3 parameters, which is far less than the typical voxel representation with a size of 32^3. This low-dimensionality is presumably beneficial for shape estimation.

The generation function $G(x)$ is trained using silhouette loss $\mathcal{L}_{\text{sl}}$ and smoothness loss $\mathcal{L}_{\text{sm}}$. Silhouette loss represents how much the reconstructed silhouettes $\{\hat{s}_i\}$ differ from the

correct silhouettes $\{s_i\}$. Smoothness loss represents how smooth the surfaces of a mesh are and acts as a regularizer. The objective function is a weighted sum of these two loss functions $\mathcal{L} = \lambda_{\text{sl}}\mathcal{L}_{\text{sl}} + \lambda_{\text{sm}}\mathcal{L}_{\text{sm}}$.

Let $\{s_i\}$ and $\{\hat{s}_i\}$ be binary masks, θ_i be the angle between two faces including the i-th edge in $G(x)$, $\mathcal{E}$ be the set of all edges in $G(x)$, and $\odot$ be an element-wise product. We define the loss functions as:

$$\mathcal{L}_{\text{sl}}(x|\phi_i, s_i) = -\frac{|\hat{s}_i \odot s_i|_1}{|\hat{s}_i + s_i - \hat{s}_i \odot s_i|_1}. \quad (5)$$

$$\mathcal{L}_{\text{sm}}(x) = \sum_{\theta_i \in \mathcal{E}} (\cos\theta_i + 1)^2. \quad (6)$$

$\mathcal{L}_{\text{sl}}$ corresponds to a negative intersection over union (IoU) between the true and reconstructed silhouettes. $\mathcal{L}_{\text{sm}}$ ensures that intersection angles of all faces are close to 180 degrees.

We assume that the object region in an image is segmented via preprocessing in common with the exiting works [5, 31, 36]. We input the mask of the object region into the generator as an additional channel of an RGB image.

4.2. Gradient-based 3D mesh editing

Gradient-based image editing techniques [6, 18] generate an image by minimizing a loss function $\mathcal{L}(x)$ on a 2D image x via gradient descent. In this work, instead of generating an image, we optimize a 3D mesh m consisting of vertices $\{\boldsymbol{v}_i\}$, faces $\{\boldsymbol{f}_i\}$, and textures $\{\boldsymbol{t}_i\}$ based on its rendered image $R(m|\phi_i)$.

4.2.1 2D-to-3D style transfer

In this section, we propose a method to transfer the style of an image x^s onto a mesh m^c.

For 2D images, style transfer is achieved by minimizing *content loss* and *style loss* simultaneously [6]. Specifically, content loss is defined using a feature extractor $f_c(x)$ and content image x^c as $\mathcal{L}_c(x|x^c) = |f_c(x) - f_c(x^c)|_2^2$. Style loss is defined using another feature extractor $f_s(x)$ and style image x^s as $\mathcal{L}_s(x|x^s) = |M(f_s(x)) - M(f_s(x^s))|_F^2$. $M(x)$ transforms a vector into a Gram matrix.

In 2D-to-3D style transfer, content is specified as a 3D mesh m^c. To make the shape of the generated mesh similar to that of m^c, assuming that the vertices-to-faces relationships $\{\boldsymbol{f}_i\}$ are the same for both meshes, we redefine content loss as $\mathcal{L}_c(m|m^c) = \sum_{\{\boldsymbol{v}_i, \boldsymbol{v}_i^c\} \in (m, m^c)} |\boldsymbol{v}_i - \boldsymbol{v}_i^c|_2^2$. We use the same style loss as that in the 2D application. Specifically, $\mathcal{L}_s(m|x^s, \phi) = |M(f_s(R(m, \phi))) - M(f_s(x_s))|_F^2$. We also use a regularizer for noise reduction. Let $\mathcal{P}$ denote the a set of colors of all pairs of adjacent pixels in an image $R(m, \phi)$. We define this loss as $\mathcal{L}_t(m|\phi) = \sum_{\{\boldsymbol{p}_a, \boldsymbol{p}_b\} \in \mathcal{P}} |\boldsymbol{p}_a - \boldsymbol{p}_b|_2^2$.

The objective function is $\mathcal{L} = \lambda_c\mathcal{L}_c + \lambda_s\mathcal{L}_s + \lambda_t\mathcal{L}_t$. We set an initial solution of m as m^c and minimize $\mathcal{L}$ with respect to $\{\boldsymbol{v}_i\}$ and $\{\boldsymbol{t}_i\}$.

4.2.2 3D DeepDream

Let $f(x)$ be a function that outputs a feature map of an image x. For 2D images, a DeepDream of image x_0 is achieved by minimizing $-|f(x)|_F^2$ via gradient descent starting from $x = x_0$. Optimization is halted after a few iterations. Following a similar process, we minimize $-|f(R(m, \phi))|_F^2$ with respect to $\{\boldsymbol{v}_i\}$ and $\{\boldsymbol{t}_i\}$.

5. Experiments

In this section, we evaluate the effectiveness of our renderer through the two applications.

5.1. Single image 3D reconstruction

5.1.1 Experimental settings

To compare our mesh-based method with the voxel-based approach by Yan *et al*. [36], we used nearly the same dataset as they did[1]. We used 3D objects from 13 categories in the ShapeNetCore [1] dataset. Images were rendered from 24 azimuth angles with a fixed elevation angle, under the same camera setup, and lighting setup using Blender. The render size was 64×64 pixels. We used the same training, validation, and test sets as those used in [36].

We compared reconstruction accuracy between the voxel-based and retrieval-based approaches [36]. In the voxel-based approach, $G(x)$ is composed of a convolutional encoder and deconvolutional decoder. While their encoder was pre-trained using the method in Yang *et al*. [37], our network works well without any pre-training. In the retrieval-based approach, the nearest training image is retrieved using the `fc6` feature of a pre-trained VGG network [26]. The corresponding voxels are regarded as a predicted shape. Note that the retrieval-based approach uses ground truth voxels for supervision.

To evaluate the reconstruction performance quantitatively, we voxelized both the ground truth meshes and the generated meshes to compute the intersection over union (IoU) between the voxels. The size of voxels was set to 32^3. For each object in the test set, we performed 3D reconstruction using the images from 24 viewpoints, calculated the IoU scores, and reported the average score.

We used an encoder-decoder architecture for the generator $G(x)$. Our encoder is nearly identical to that of [36], which encodes an input image into a 512D vector. Our

[1]The dataset we used was not exactly the same as that used in [36]. The rendering parameters for the input images were slightly different. Additionally, while our silhouette images were rendered by Blender from the meshes in the ShapeNetCore dataset, theirs were rendered by their PTNs using voxelized data.

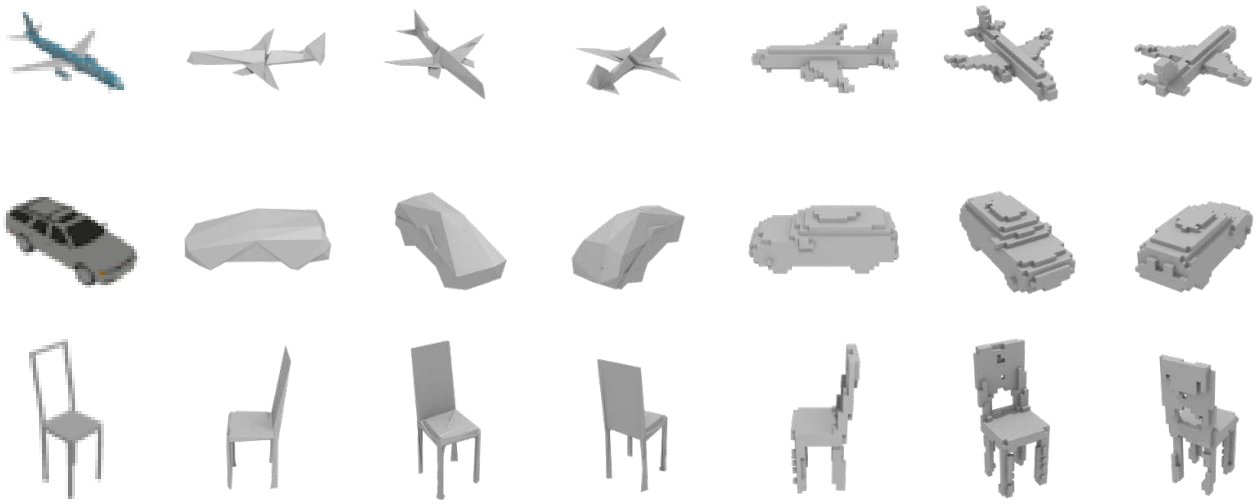

Figure 4. 3D mesh reconstruction from a single image. Results are rendered from three viewpoints. First column: input images. Second through fourth columns: mesh reconstruction (proposed method). Fifth through seventh columns: voxel reconstruction [36].

	airplane	bench	dresser	car	chair	display	lamp
Retrieval [36]	0.5564	0.4875	0.5713	0.6519	0.3512	0.3958	0.2905
Voxel-based [36]	0.5556	0.4924	0.6823	**0.7123**	0.4494	0.5395	**0.4223**
Mesh-based (ours)	**0.6172**	**0.4998**	**0.7143**	0.7095	**0.4990**	**0.5831**	0.4126

	loudspeaker	rifle	sofa	table	telephone	vessel	mean
Retrieval [36]	0.4600	0.5133	0.5314	0.3097	0.6696	0.4078	0.4766
Voxel-based [36]	0.5868	0.5987	0.6221	**0.4938**	0.7504	0.5507	0.5736
Mesh-based (ours)	**0.6536**	**0.6322**	**0.6735**	0.4829	**0.7777**	**0.5645**	**0.6016**

Table 1. Reconstruction accuracy measured by voxel IoU. Higher is better. Our mesh-based approach outperforms the voxel-based approach [36] in 10 out of 13 categories.

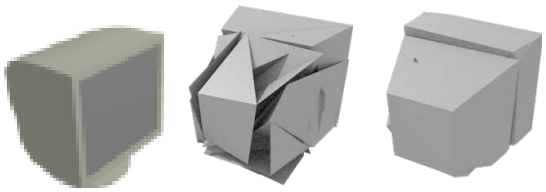

Figure 5. Generation of the back side of a CRT monitor with/without smoothness regularizer. Left: input image. Center: prediction without regularizer. Right: prediction with regularizer.

decoder is composed of three fully-connected layers. The sizes of the hidden layer are 1024 and 2048.

The render size of our renderer is set to 128×128 and downsampled them to 64×64. We rendered only the silhouettes of objects without using textures and lighting. We set $\lambda_{\text{sl}} = 1$ and $\lambda_{\text{sm}} = 0.001$ in Section 5.1.2, and $\lambda_{\text{sm}} = 0$ in Section 5.1.3. We trained our generator using the Adam optimizer [11] with $\alpha = 0.0001$. The batch size was set to 64. In each minibatch, we included silhouettes from two viewpoints per input image.

5.1.2 Qualitative evaluation

We trained 13 models with images from each class. Figure 4 presents a part of results from the test set by our mesh-based method and the voxel-based method [36][2]. Additional results are presented in the supplementary materials. These results demonstrate that a mesh can be correctly reconstructed from a single image using our method.

Compared to the voxel-based approach, the shapes reconstructed by our method are more visually appealing from the two points. One is that a mesh can represent small parts, such as airplane wings, with high resolution. The other is that there is no cubic artifacts in a mesh. Although low resolutions and artifacts may not be a problem in tasks such as picking by robots, they are disadvantageous for computer graphics, computational photography, and data augmentation.

Without using the smoothness loss, our model sometimes produces very rough surfaces. That is because the

[2]We trained generators using the code from the authors and our dataset.

smoothness of surfaces has little effect on silhouettes. With the smoothness regularizer, the surface becomes smoother and looks more natural. Figure 5 illustrates the effectiveness of the regularizer. However, if the regularizer is used, the voxel IoU for the entire dataset becomes slightly lower.

5.1.3 Quantitative evaluation

We trained a single model using images from all classes. The reconstruction accuracy is shown in Table 1. Our mesh-based approach outperforms the voxel-based approach [36] for 10 out of 13 categories. Our result is significantly better for the `airplane`, `chair`, `display`, `loudspeaker`, and `sofa` categories. The basic shapes of the `loudspeaker` and `display` categories are simple. However, the size and position vary depending on the objects. The fact that meshes are suitable for scaling and translation presumably contributes to the performance improvements in these categories. The variations in shapes in the `airplane`, `chair` and `sofa` categories are also relatively small.

Our approach did not perform very well for the `car`, `lamp`, and `table` categories. The shapes of the objects in these categories are relatively complicated, and they are difficult to be reconstructed by deforming a sphere.

5.1.4 Limitation

Although our reconstruction method already surpasses the voxel-based method in terms of visual appeal and voxel IoU, it has a clear disadvantage in that it cannot generate objects with various topologies. In order to overcome this limitation, it is necessary to generate the faces-to-vertices relationship $\{\boldsymbol{f}_i\}$ dynamically. This is beyond the scope of this study, but it is an interesting direction for future research.

5.2. Gradient-based 3D editing via 2D loss

5.2.1 Experimental settings

We applied 2D-to-3D style transfer and 3D DeepDream to the objects shown in Figure 6. Optimization was conducted using the Adam optimizer [11] with $\beta_1 = 0.9$, and $\beta_2 = 0.999$. We rendered images of size 448×448 and downsampled them to 224×224 to eliminate aliasing. The batch size was set to 4. During optimization, images were rendered at random elevations and azimuth angles. Texture size was set to $s_t = 4$.

For style transfer, the style images we used were selected from [3, 9]. λ_c, λ_s, and λ_t are manually tuned for each input. The feature extractors f_s for style loss were `conv1_2`, `conv2_3`, `conv3_3`, and `conv4_3` from the VGG-16 network [26]. The intensities of the lights were $l^a = 0.5$ and $l^d = 0.5$, and the direction of the light was randomly set during optimization. The α value of Adam was set to $2.5\text{e}{-4}, 5\text{e}{-2}$ for $\{\boldsymbol{v}_i\}, \{\boldsymbol{t}_i\}$. The number of parameter updates was set to $5,000$.

In DeepDream, images are rendered without lighting. The feature extractor was the `inception_4c` layer from GoogLeNet [28]. The α value of Adam was set to $5\text{e}{-5}, 1\text{e}{-2}$ for $\{\boldsymbol{v}_i\}, \{\boldsymbol{t}_i\}$. Optimization is stopped after $1,000$ iterations.

5.2.2 2D-to-3D Style Transfer

Figure 7 presents the results of 2D-to-3D style transfer. Additional results are shown in the supplementary materials.

The styles of the paintings were accurately transferred to the textures and shapes. From the outline of the bunny and the lid of the teapot, we can see the straight style of Coupland and Gris. The wavy style of Munch was also transferred to the side of the teapot. Interestingly, the side of the tower of Babel was transferred only to the side, not to the upside, of the bunny.

The proposed method provides a way to edit 3D models intuitively and quickly. This can be useful for rapid prototyping for product design as well as art production.

5.2.3 3D DeepDream

Figure 8 presents the results of DeepDream. A nose and eyes emerged on the face of the bunny. The spout of the teapot expanded and became the face of the bird, while the body appeared similar to a bus. These transformations matched the 3D shape of each object.

6. Conclusion

In this paper, we enabled the integration of rendering of a 3D mesh into neural networks by proposing an approximate gradient for rendering. Using this renderer, we proposed a method to reconstruct a 3D mesh from a single image, the performance of which is superior to the existing voxel-based approach [36] in terms of visual appeal and the voxel IoU metric. We also proposed a method to edit the vertices and textures of a 3D mesh according to its 3D shape using a loss function on images and gradient descent. These applications demonstrate the potential of integrating mesh renderers into neural networks and the effectiveness of the proposed renderer.

The applications of our renderer are not limited to those presented in this paper. Other problems will be solved through incorporating our module in other systems.

Acknowledgment

This work was partially funded by ImPACT Program of Council for Science, Technology and Innovation (Cabinet Office, Government of Japan) and partially supported by JST CREST Grant Number JPMJCR1403, Japan.

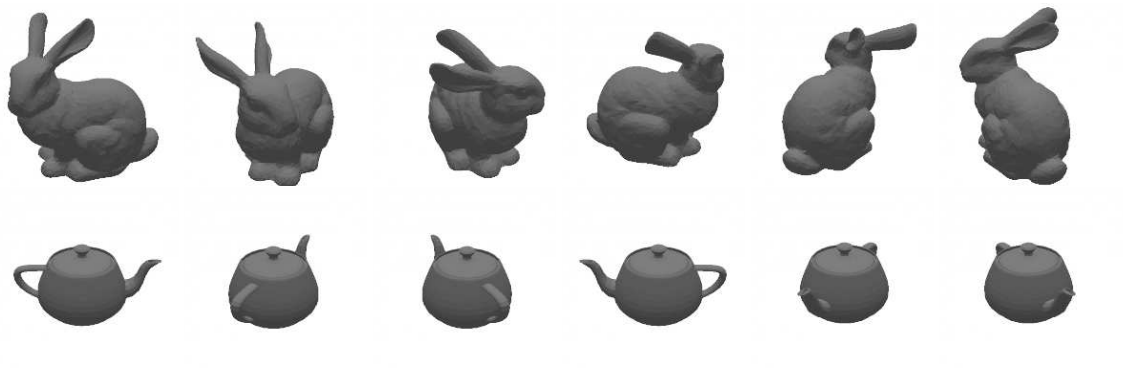

Figure 6. Initial state of meshes in style transfer and DeepDream. Rendered from six viewpoints.

Figure 7. 2D-to-3D style transfer. The leftmost images represent styles. The style images are *Thomson No. 5 (Yellow Sunset)* (D. Coupland, 2011), *The Tower of Babel* (P. Bruegel the Elder, 1563), *The Scream* (E. Munch, 1910), and *Portrait of Pablo Picasso* (J. Gris, 1912).

Figure 8. DeepDream of 3D mesh.

References

[1] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv*, 2015. 5

[2] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *ECCV*, 2016. 1, 2

[3] V. Dumoulin, J. Shlens, M. Kudlur, A. Behboodi, F. Lemic, A. Wolisz, M. Molinaro, C. Hirche, M. Hayashi, E. Bagan, et al. A learned representation for artistic style. *ICLR*, 2017. 7

[4] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, 2014. 2

[5] H. Fan, H. Su, and L. Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, 2017. 2, 5

[6] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *CVPR*, 2016. 2, 3, 4, 5

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014. 4

[9] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 7

[10] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri. 3d shape segmentation with projective convolutional networks. *CVPR*, 2017. 2

[11] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6, 7

[12] R. Klokov and V. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *ICCV*, 2017. 2

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[14] J. Li, K. Xu, S. Chaudhuri, E. Yumer, H. Zhang, and L. Guibas. Grass: Generative recursive autoencoders for shape structures. In *SIGGRAPH*, 2017. 2

[15] M. M. Loper and M. J. Black. Opendr: An approximate differentiable renderer. In *ECCV*, 2014. 2

[16] S. Marschner and P. Shirley. *Fundamentals of computer graphics*. CRC Press, 2015. 2, 3, 4

[17] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS*, 2015. 1, 2

[18] A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks. *Google Research Blog*, 2015. 2, 3, 4, 5

[19] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 2

[20] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *CVPR*, 2016. 1, 2

[21] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 2

[22] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016. 4

[23] D. J. Rezende, S. A. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, and N. Heess. Unsupervised learning of 3d structure from images. In *NIPS*, 2016. 2

[24] G. Riegler, A. O. Ulusoys, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *CVPR*, 2017. 1, 2

[25] A. Saxena, S. H. Chung, and A. Y. Ng. 3-d depth reconstruction from a single still image. *IJCV*, 76(1):53–69, 2008. 2

[26] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015. 5, 7

[27] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*, 2015. 2

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 7

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

[30] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *ICCV*, 2017. 1, 2

[31] S. Tulsiani, T. Zhou, A. A. Efros, and J. Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *CVPR*, 2017. 1, 2, 5

[32] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. In *SIGGRAPH*, 2017. 2

[33] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 2

[34] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NIPS*, 2016. 1, 2

[35] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. 1, 2

[36] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *NIPS*, 2016. 1, 2, 4, 5, 6, 7

[37] J. Yang, S. E. Reed, M.-H. Yang, and H. Lee. Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. In *NIPS*, 2015. 5

[38] L. Yi, H. Su, X. Guo, and L. Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *CVPR*, 2017. 2

[39] C. Zou, E. Yumer, J. Yang, D. Ceylan, and D. Hoiem. 3d-prnn: Generating shape primitives with recurrent neural networks. In *ICCV*, 2017. 2