Deep Convolution Neural Networks for Painting-like 3D Rendering

Zhi Yang[1], Pei-Li Sun[1], Tzung-Han Lin[1]

[1]National Taiwan University of Science and Technology, Taipei 10607, Taiwan
Keywords: Deep learning, non-photorealistic rendering, computer graphics.

ABSTRACT

A 3D rendering model which uses deep convolutional neural networks to imitate 2D painting style is proposed. User can feed the networks with simple paintings of specific objects to render images of 3D objects with any orientations in accordance with the painting style.

1 INTRODUCTION

Today, 3D rendering technology is widely used in 3D animation, video games and artwork, etc. In the 3D rendering pipeline, shading is the final step [1]. The calculation of shading involves many information of the 3D scene, such as lighting and viewing geometry, normal vectors and optical property of object surface, and colors to generate the RGB values on each pixel of the final 2D rendering image. In addition to the rendering methods which use the physical lighting model to render the photorealistic images (physically based rendering, PBR), there are many other methods [2][3] using different process to achieve non-photorealistic rendering (NPR). We can use professional software or write our own programs to create the desired rendering style. However, if a user doesn't have the knowledge of physical lighting and rendering style, it is very difficult to create a rendering method to generate the desired NPR images. This study uses deep convolutional neural networks (DCNNs) to solve this program.

In recent years, convolutional neural networks (CNN) has been widely used for pattern recognition. Some advanced CNNs can even generate per-pixel results, such as object segmentation [4]. In terms of 3D information processing, Nalbach et al. [5] used CNN to render photorealistic images, and Taniai and Maehara [6] applied CNN to estimate photometric stereo. In addition to the superior multi-scale feature extraction and nonlinear fitting of CNN, smart learning also is needed in the non-photorealistic rendering. Using the DCNNs, we don't need to know the rules and the math of the desired rendering style. All we need is to directly define the ideal results, that means we can provide handmade training samples, and DCNNs can learn how to render a 3D scene like a human painter.

2 EXPERIMENT

In our proposed rendering pipeline (**Fig. 1**), we first render pixel-wise 3D information (including object mask, depth and normal vectors) and illumination information (include diffuse, specular and light directions) of the 3D models to be painted, then the information will be processed by the trained DCNNs to generate the painting-like 2D outputs.

In the training phase of the DCNNs, the input data are the 3D and illumination information of specific (sphere-like) models. The use must paint 2D image of the specific (sphere-like) models in different illumination directions. These 2D paintings are the ground truths (i.e., output targets) of the training data. The DCNNs will learn how to imitate the 2D painting style based on the 3D and illumination information of the specific models.

This section will introduce more details about how to prepare the training data, to generate more training set, DCNNs architecture, and the rendering methods.

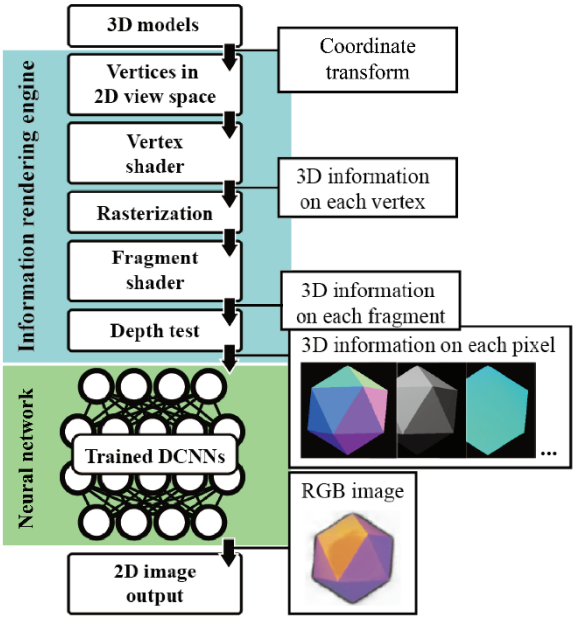

Fig.1 The proposed rendering pipeline.

2.1 Prepare training data

The specific 3D models which user need to paint are smooth sphere, polygonal sphere and mesh pattern sphere **(Fig. 2)**. We choose the smooth sphere because it contains complete and continuous normal vectors. To learn how to paint discontinuous surfaces, such as edges, corners and cracks (**Fig. 3**) using the pipeline, we use polygonal sphere and mesh pattern sphere.

The user must paint the appearance of seven different

illumination directions of the smooth sphere with the same painting style. The configuration of the 7 point light-sources is shown in **Fig. 4**. We place the light sources around the model from 0 to 180° polar angle, and the angle between two neighboring light sources is 30°. For clarity, we named each light source as $L_{(\theta, \varphi)}$, θ is azimuthal angle and φ is polar angle. The purpose of this setting is that we can generate great amount of different illumination directions by these seven light sources, more details will be introduced in Section 2.2. Polygonal sphere and mesh pattern sphere are only used to learn how to paint the discontinuous surfaces, so we can simply choose the light source $L_{(0°, 30°)}$.

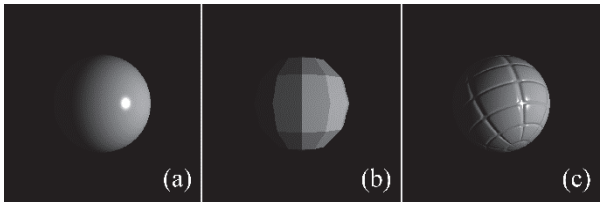

Fig.2 3D models we use to create the training set: (a) Smooth sphere, (b) Polygonal sphere, (c) Mesh pattern sphere.

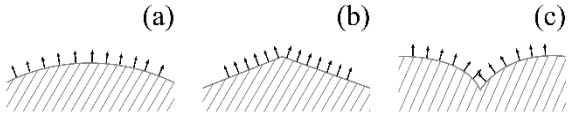

Fig.3 Different types of surface: (a) Smooth surface, (b) Edge or corner, (c) Crack.

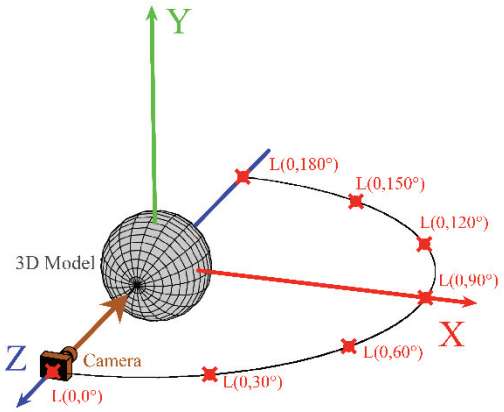

Fig.4 Configuration of point light sources.

The 3D and illumination information, which is the input of our training data, is generated by our rendering engine, which was built by C++ and OpenGL. The 3D information includes object masks, depth maps and normal vectors; and the illumination information includes diffuse, specular and light directions. The diffuse and specular are generated by Phong shading. The information mentioned above is usually used as the reference when we are painting.

2.2 Pre-process the training data

To enhance performance of our DCNNs, we generating more training sets by rotating the 2D painting images. The painting images of smooth sphere in 5 different light sources including $L_{(0°,30°)}$, $L_{(0°,60°)}$, $L_{(0°,90°)}$, $L_{(0°,120°)}$ and $L_{(0°,150°)}$ are rotated two degrees step by step. This process will produce 902 ground truth images of smooth sphere in different illumination directions. And for both polygonal sphere and mesh pattern sphere, that will produce 180 ground truth images.

The 3D and illumination information corresponding to these newly generated ground truth images can be easily produced by changing the position of the light source and transforming the model in the OpenGL program.

Finally, we got 1262 training sets; we can start the DCNNs training.

2.3 Architecture of DCNNs

The proposed DCNNs is built with Tensorflow. We use U-shape neural networks as the DCNNs architecture (**Fig. 5**). The structure can produce pixel-wise outputs. The whole structure has two paths (see **Fig. 5**): left one is contracting path and right one is expansive path. The size of each input image is 512x512 pixels. We stack these images and feed this 512x512x10 (depth of normal vectors and light directions are both in 3 channels and the other are 1) input data into the contracting path. The resolution of feature map will be decreased and the depth will be increased through each step. One step contains two convolutions and ReLU layers, we do down-sampling on the output feature maps of each step through pooling and then pass it to the next step. The feature map is contracted to the size of 32x32x1024, then enter the expansive path. Steps in the expansive path will increase the resolution and decrease the depth. The output of each step in this path will be up-sampling through up-convolution and passed to the next step. Before it enters the next step, it will concatenate with the feature maps from the same step in the contracting path first. In the last step of the expansive path, the network finally generates the result, which is a 512x512 resolution RGB image, through two convolutions and ReLU layers.

We set the size of each convolution filter to be 3x3, and the size of each step is shown in **Fig. 5**. We use PSNR (peak signal-to-noise ratio) as the loss function of our network, and use Adam to optimize our model.

In addition to the pixel-wise output of this architecture, it also has a benefit that the loss of high frequency details caused by the down-sampling will be reduced. It is because of the concatenation between the contracting and the expansive paths.

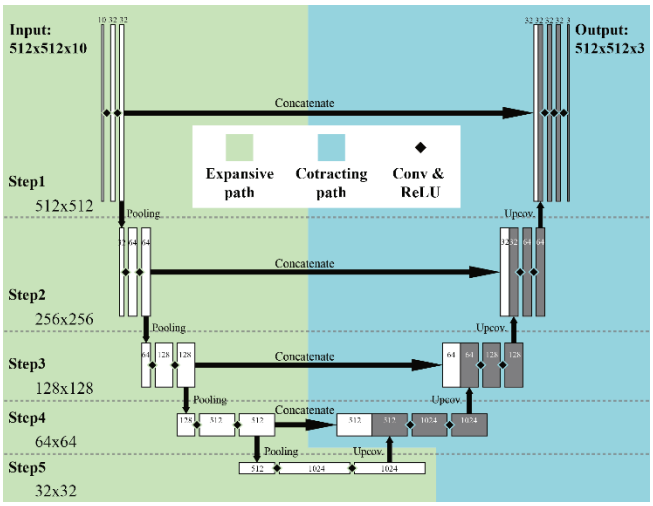

Fig. 5 Networks architecture

2.4 Generate 3D and illumination information

Our input information contains object mask, depth, normal vectors, diffuse, specular and light direction, as show in **Fig. 6**. We create some shaders to generate the information, and there are some details about processing the normal vectors, diffuse and specular reflections of a surface.

2.5 Normal vectors

The normal vectors were stored in the 3D models, so we do not have to generate it by ourselves. All we need to do is to transform the coordinate system of normal vectors from world space to camera space. It is easy to do by multiply the normal vectors with the model matrix and view matrix. However, normal vectors are only direction vectors and do not represent a specific position in a space. If we want to multiply the normal vectors with these matrices, we should set the fourth component of a normal vector to zero, and then we can multiply with the 4x4 matrix without translation. **Equ. 1** shows the coordinate transformation of normal vectors.

$$\begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \\ 0 \end{bmatrix} = M_{view} \cdot M_{model} \cdot \begin{bmatrix} x_{local} \\ y_{local} \\ z_{local} \\ 0 \end{bmatrix} \quad (1)$$

The subscripts “local” denote the local space and “cam” denote the camera space. After this process, we can simply get the normal vectors in the camera space by normalizing the result.

2.6 Diffuse & specular

We use Phong reflection model (**Equ. 2**) to calculate diffuse and specular light reflections of the 3D model. The parameter setting is shown in **Table 1.** Where the $\hat{L}$, $\hat{N}$, $\hat{R}$ and $\hat{V}$ represent vectors of lighting, surface normal, surface reflection and viewing respectively, and the subscript *m* is an index number for different light source. We generate diffuse reflection by setting the i_s in **Equ. 2** as zero, and generate specular reflection by setting both of i_a and i_d equals to zero.

$$I_p = K_a i_a + \sum_{m \in lights} \left(K_d (\hat{L}_m \cdot \hat{N}) i_{m,d} + K_s (\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,s} \right) \quad (2)$$

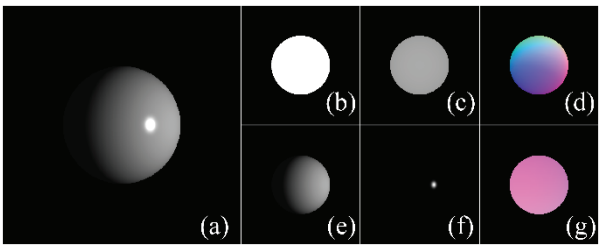

Fig. 6 Input information of one training set: (a) Phong shading, (b) object mask, (c) depth (z co-ordinates), (d) normal vector, (e) diffuse, (f) specular and (g) light direction.

Table 1 Parameter setting in Equ. 2

Symbol	Describe	Setting
i_a	Ambient light	R= 0.2, G= 0.2, B= 0.2
i_d	Diffuse light	R= 0.8, G= 0.8, B= 0.8
i_s	Specular light	R= 1.0, G= 1.0, B= 1.0
K_a	Ambient reflection constant	R= 0.2, G= 0.2, B= 0.2
K_d	Diffuse reflection constant	R= 0.8, G= 0.8, B= 0.8
K_s	Specular reflection constant	R= 1.0, G= 1.0, B= 1.0
α	Shininess	64

3 RESULTS

After training the proposed DCNNs, we can finally render images by our rendering pipeline. We use two datasets to train our DCNNs, one is painted with watercolor, another is painted with computer painting software. The paintings of each dataset are shown in **Fig.7**. To demonstrate the rendering ability of our pipeline, we chose six 3D models as our input, the rendering result is shown in **Fig. 8** and **Fig. 9**.

We could see that the DCNNs can rendering color correctly with both painting style in various light directions.

Our pipeline can also paint outline of each model, and for **Fig. 9**, if you look carefully, you could find out that the outlines have different width due to the light direction just like the paintings in the dataset.

However, the DCNNs have limitation in learning some painting details. For the watercolor dataset, DCNNs fail to learn some properties of watercolor, like water stain and uneven colors. For the software dataset, DCNNs learn to paint the edge between two different colors, but the edge is not as sharp as the original painting. For the both datasets, the polygonal sphere and mesh pattern

sphere are prepared to let DCNNs learn to draw inner contours, but it fail to generate the feature.

Although the results of both painting styles are a little bit blur and smoother than real paintings, in general, our pipeline can paint the 3D object with correct color and render the image with similar looking as the given painting style.

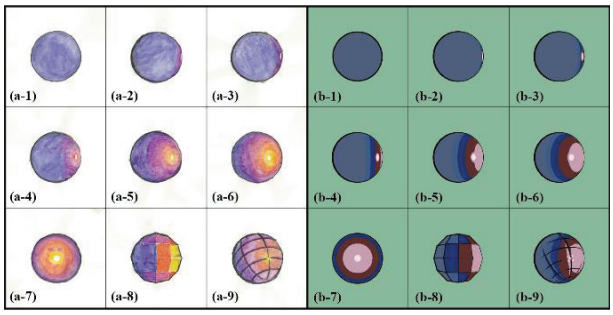

Fig. 7 Ground truths of the training data: (a) Paint with watercolor, (b) paint with painting software. The corresponding light source of 1-7 are $L_{(0,\,180°)}$-$L_{(0,\,0)}$, and light source of 8 and 9 are $L_{(0,\,30°)}$

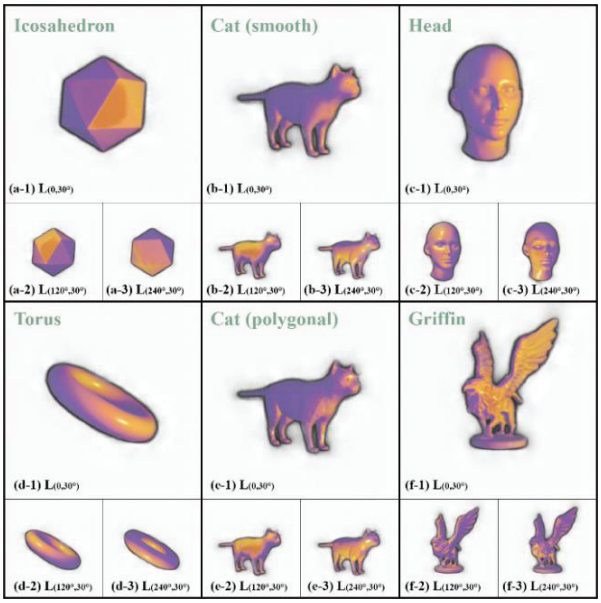

Fig. 8 Test results of the watercolor dataset.

4 CONCLUSIONS

To imitate a human painting style and to apply it in 3D rendering, we propose a rendering pipeline, which combines a rendering engine and DCNNs. The trained DCNNs use the information which generated by our rendering engine to render the final paintings with specific painting style. To ensure we have enough dataset to train our DCNNs, we provide a procedure which can use only nine human paintings to generate a great number of training data. After the training, this pipeline can render images of 3D objects with any light orientations in accordance with the painting style.

In the future, we consider adding more 3D and illumination information to include more effects, like shadow and particle effects. And we also consider to combine two or more different rendering styles altogether into our pipeline

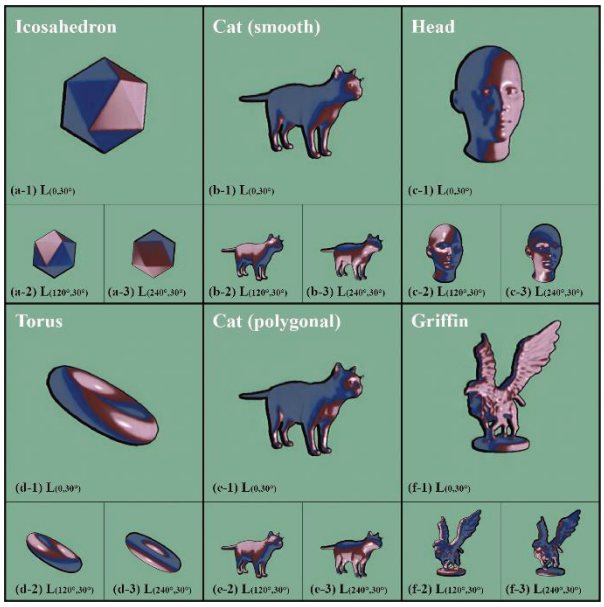

Fig. 9 Test results of the painting software dataset.

REFERENCES

[1] R. J. Rost, B. Licea-Kane, D. Ginsburg, J. Kessenich, B. Lichtenbelt, H. Malan, and M. Weiblen, *OpenGL Shading Language*, 3rd ed, Michigan: Addison Wesley, (2009).

[2] S. M. F. Treavett and M. Chen, “Pen-and-Ink Rendering in Volume Visualization”, *Proceedings of Visualization 2000*, VIS 2000, (2000).

[3] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer, and D. H. Salesin, “Computer-Generated Watercolor”, *SIGGRAPH '97*, (1997).

[4] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation”, *MICCAI 2015*, (2015).

[5] O. Nalbach, E. Arabadzhiyska, D. Mehta, H.-P. Seidel, and T. Ritschel, “Deep Shading: Convolutional Neural Networks for Screen Space Shading”, *Computer Graphic Forum*, vol. 36, no. 4, pp. 65-78, (2017).

[6] T. Taniai and T. Maehara, “Neural Inverse Rendering for General Reflectance Photometric Stereo, *ICML 2018*, (2018).