An Automatic Texture Generation Algorithm for 3D Shapes Based on cGAN

Xiao Huang¹, Yigang Wang¹, * and Zizhao Wu¹
¹Hangzhou Dianzi University, Hangzhou 310018, China

* yigang.wang@hdu.edu.cn

Abstract. Texturing 3D shapes is of great importance in computer graphics with applications ranging from game design to augmented reality. However, the processes of texture generation are usually tedious, time-consuming and labor-intensive. In this paper, we propose an automatic texture generation algorithm for 3D shapes based on conditional Generative Adversarial Networks (cGAN). The core of our algorithm includes sampling the model outline and building a cGAN in order to generate model textures automatically. In particular, we propose a novel edge detection method using 3D model information which can accurately find the outline of the model to improve the quality of the generated texture. Due to the adaptability of the algorithm, our approach is suitable for texture generation for most 3D models. Experimental results show the efficiency of our algorithm which can easily generate high quality model textures.

1. Introduction

3D model texture plays a crucial role in computer graphics. Currently, there are a large number of texturing technologies in existence. However, the process of creating texture is usually tedious and labor-intensive. For example, most of the texture are sampled in real-world, and then processed by a variety of image algorithms. The whole process is time-consuming. Therefore, the current process of texture production is very complex.

With arising of deep learning, Generative Adversarial Networks (GAN)[1] was proposed which learns a mapping from noise data to textured image using a training set of images. One of its important applications is image generation[2]. However, the output of GAN is uncontrollable. To solve this problem, conditional Generative Adversarial Networks (cGAN)[3] was proposed which can use images as input of the generator. Due to the controllable output of cGAN, a variety of impressive results were achieved based on cGAN such as style transfer[4], image-to-image translation[5] and photo generation[6].

In this paper, we propose an automatic texture generation algorithm for 3D shapes based on cGAN. This algorithm can automatically generate model textures for the none-textured 3D model. Our method first presents a novel edge detection approach which can find the outline of the 3D model more accurately and efficiently. Then, combined with this edge detection approach, we build a cGAN which can translate the outline image to the coloring image to generate the model texture. Since the model texture generated by our approach are determined by the training data, the proposed algorithm can automatically infer the texture color and is effective on a wide range of 3D models. Experimental results demonstrate that our algorithm has high efficiency and visual quality.
2. Related Work
The algorithm proposed in this paper includes edge detection and image-to-image translation, which we discuss in the following.

2.1. Edge detection
Edge detection is the basic method of computer vision aims to identify points in the image where the brightness changes greatly. This makes edge detection can eliminate redundant information in the image based on the important structural properties of the image. Edge detection plays an important role in computer vision. The classic edge detection methods are Sobel operator detection [7] and Canny operator detection [8]. With the development of Convolutional Neural Networks (CNN), edge detection methods can gradually learn image features automatically, including Deep-Contour [9] and CSCNN [10]. But these methods are all based on image information. Edge detection to the sampled image of the three-dimensional model is still affected by environmental factors such as illumination and shadow, which lead to many redundant edge information. Based on the above observation, we suggest an edge detection method by utilizing the 3D model information to facilitate the texture generation process.

2.2. Image-to-image translation
With the increasing variety of image generation methods, Hertzmann et al. [11] proposed the idea of image-to-image translation. With a wave of development using GAN, Isola et al. [12] proposed the pix2pix algorithm based on cGAN to generate images. Because cGAN has the characteristics of adding constraints to the generator and discriminator to ensure the controllability of the output, which has a better performance in solving the image-to-image translation problem. However, most of the previous research works focus their attention on image processing, and there are few related researches has been paid their attention to the domain of 3D models. Based on the characteristics of pix2pix for image translation at pixel level, we use this algorithm to color the wireframe texture.

3. Method
Our automatic texture generation algorithm transforms the none-texture model into a textured model through four steps: model sampling, edge detection, texture coloring and texture mapping. The procedure of our algorithm is as shown in Figure 1.

![Figure 1](image.png)

Figure 1. The procedure of our 3D model texture generation algorithm.

3.1. Model Sampling
The first step of generating a model texture is to transform the model information defined in 3D domain into image information in 2D domain. Firstly, the relevant parameters of the rendering pipeline camera, including position, up vector, field angle of view, near plane distance and far plane distance, need to be set during the sampling process to ensure that the sampled picture conforms to the input standard of cGAN. After specifying the camera parameters, the frame buffer content is rendered into a texture with the same size to the camera's viewport, as shown in Figure 2.
3.2. Edge detection

In order to eliminate the interference of illumination, such as shadow and material color on edge detection, we use an edge detection algorithm based on 3D model information. The idea of the algorithm is to use the depth information of the model to discriminate the edge of the model. As a result, a deep texture will be generated based on the depth information of the model. Firstly, we get the depth information of the model vertices from the Normalized Device Coordinates (NDC) through the rendering pipeline. Secondly, the depth information of vertices is translated to the depth value of fragments by rasterization. Then, we generate a 16-bit 2-channel texture to store depth information. Finally, the depth values corresponding to all the fragments are encoded into the two channels of the depth texture.

The position of an edge is determined by calculating the depth difference in the diagonal direction by taking account of the deep texture neighbourhood pixels. When the difference exceeds a certain threshold, it means that there is an edge between two pixels. Two parameters $D_s$ and $T_d$ are set to control the effect of edge detection, where $D_s$ is the sampling distance for the texture and $T_d$ is the threshold for the depth difference. The larger the sampling distance of the texture represents the wider edge. The larger the threshold of the depth difference represents the more edges that can be detected. The specific algorithm steps are as follows:

1. According to sampling distance $D_s$, the depth value matrix $D_d$ of the domain pixels is sampled from the depth texture.

   \[
   D_d = \begin{bmatrix}
   d_{11} & d_{12} \\
   d_{21} & d_{22}
   \end{bmatrix}
   \]

   Where $d_{ij}(i \in [1,2], j \in [1,2])$ is the depth value of the corresponding pixel $i$ and $j$.

2. Since the depth value behind the projection transformation is nonlinear, the depth value $d_{ij}$ needs to be mapped to a linear depth value $z_{ij}(i \in [1,2], j \in [1,2])$ under the view space. The formula is defined to be:

   \[
   z_{ij} = \frac{1}{s_n s_f (d_{ij} + 1) / s_n}
   \]

   where $s_n$ is the near plane distance of the camera frustum, $s_f$ is the far plane distance for the camera frustum.

3. The depth differences $T_1$ and $T_2$ are obtained by calculating the diagonal difference of the depth value in the four neighborhood pixels.

4. Compare the size of $T_1$ and $T_2$ with thresholds $T_d$. If both of them exceed the threshold $T_d$, indicating that the depth difference is large, we argue that there is an edge between the two pixels. Otherwise, there is no edge between the two pixels.

5. Loop through the above steps until traversing the entire depth texture, and finally we get an outline of the model.
3.3. Texture Coloring

We build a generator G and a discriminator D to achieve texture coloring. The aim of the generator G is to translate the input outline image to the realistic-looking color image, while the discriminator D is used to distinguish the ground truth images from the translated images. The dataset of training is a set of picture pairs \( \{a_i, b_i\} \), where \( a_i \) is the outline images and \( b_i \) is the corresponding color image.

The loss function of the network is as follow:

\[
\mathcal{L}_{cGAN} = \mathbb{E}_{(a,b)}[\log D(a,b)] + \mathbb{E}_a[\log(1 - D(a,G(a)))]
\]  

(3)

The objective of generator G is to minimize the loss function value, and discriminator D aims to maximize it. The generator G consists of eight convolution layers and eight deconvolution layers, as shown in Figure 3. The discriminator D consists of five convolutional layers, as shown in Figure 4.

![Figure 3. Architecture of our generator G, which aims to translate the outline images to the color images.](image)

![Figure 4. Architecture of our discriminator D, which aims to distinguish the real color images from translated ones.](image)

The specific algorithm steps are as follows:

1. Acquire the picture pairs \( \{a_1, b_1\}, \{a_2, b_2\}, ..., \{a_n, b_n\} \) from the dataset, where \( b_i \) is the real color image, \( a_i \) is the outline image of the \( b_i \) and \( n \) is the number of all picture pairs.

2. By fed \( a_i \) and \( b_i \) into the discriminator D, we can get the output \( D(a, b) \).

3. The generator G translates the outline image \( a_i \) to the three-channel color image \( G(a) \).

4. By taking the combination of \( a_i \) and \( G(a) \) as input to the discriminator D, we obtain the output \( D(a,G(a)) \).

5. Train all pairs of images in the dataset by the gradient descent algorithm. The loss functions of the discriminator D and generator G are as follow:

\[
\mathcal{L}_D = \mathbb{E}_{a,b} [\log D(a,b)] + \mathbb{E}_a[\log(1 - D(a,G(a)))]
\]  

(4)

\[
\mathcal{L}_G = \mathbb{E}_a \left[ \log \left( D(a,G(a)) \right) \right] + \lambda \mathcal{L}_{L1}(G)
\]  

(5)

\[
\mathcal{L}_{L1}(G) = \mathbb{E}_{a,b} [\|b - G(a)\|_1]
\]  

(6)

where \( \mathcal{L}_{L1}(G) \) is L1 distance.

6. After completing the training, the outline image of the 3D model is taken as input the generator G to produce the output of the colored texture.
3.4. Texture Mapping

Accurate mapping the texture to the model is achieved through a series of coordinate transformations. The specific algorithm steps are as follows:

1. The render camera parameters are restored for the correct calculation of the texture coordinates.
2. For calculating the texture coordinates \((u, v)\) corresponding to the vertex coordinate \(P_0 (x_0, y_0, z_0, w_0)\), we use the model transformation matrix \(M_{model}\), the view transformation matrix \(M_{view}\) and the clip transformation matrix \(M_{clip}\) to convert the vertex coordinate \(P_0\) to the \(P_c (x_c, y_c, z_c, w_c)\) which is vertex coordinate in the clip space. The transformation formula is as follows:

\[
P_c = M_{clip} M_{view} M_{model} P_0
\]

3. The \(x\) component of \(P_c\) and the \(y\) component of \(P_c\) are sequentially divided by the \(w\) component of \(P_c\) to obtain the coordinates \((\frac{x_c}{w_c}, \frac{y_c}{w_c})\) under the NDC. The range of \(\frac{x_c}{w_c}\) and \(\frac{y_c}{w_c}\) are \([-1, 1]\), but the range of texture coordinates \(u\) and \(v\) are \([0, 1]\). For acquiring the correct texture coordinates, \(\frac{x_c}{w_c}\) and \(\frac{y_c}{w_c}\) are transformed by the following formulas.

\[
u = \frac{x_c}{2w_c} + \frac{1}{2}
\]

\[
v = \frac{y_c}{2w_c} + \frac{1}{2}
\]

4. By sampling from the generated texture image according to the texture coordinates \((u, v)\), the color value of the pixel is obtained. Then we integrate the color value into the framebuffer.

5. Loop through the above steps until traversing all vertices to get a 3D model with texture.

4. Results

To verify the validity of our algorithm, we evaluate it from two aspects which are the texture visual quality generated based on different models and different edge detection algorithms. We used the UT Zappos50K dataset which contains 50k images to test the method in this paper.

4.1. The visual quality of textures generated based on different models

For the image generation problem, the plausibility of a human observer is an important criterion for judging the quality of the generated image. The higher plausibility represents the higher quality of the generated image. Therefore, we use the criterion of plausibility to evaluate the visual quality of texture. The experiment sets from 1 to 10 plausibility level labels for observer selection, where '10' represents highest plausibility and '1' represents the lowest plausibility. The observer needs to complete the evaluation of 50 sets of experiments. Each set of experiments contains a none-textured model and a corresponding model with texture. In order to ensure the accuracy of the evaluation, the observer needs to complete the plausibility evaluation within three seconds in each group of comparison experiments.

![Figure 5](image)

**Figure 5.** Results of our plausibility evaluation based on different 3D models. Higher values indicate higher ratio of observers choosing that level of plausibility.
Figure 5 shows that the observer's evaluation on the plausibility of the 3D model texture generated by the algorithm is concentrated at 7, 8 and 9 levels, which is a high degree of plausibility, indicating that the 3D model texture generated by the algorithm has high plausibility. It is proved that the texture quality generated by the algorithm is better. The comparison of the texture generation effects of different models is presented in the following:

![Figure 6. Results of our algorithm on several models.](image)

Figure 6 shows that there are good visual effects for different models, which indicates that the proposed method is applicable to different models.

4.2. The visual quality of texture generated based on different edge detection algorithms

In order to verify the validity of the edge detection method in this paper, based on the criteria for generating image plausibility mentioned in 4.1, we design a comparison experiment which compares our method with the Sobel method and the Canny method. We set up 50 sets of experiments and each of them shows the models with textures generated based on three edge detection methods to make plausibility evaluation. In order to ensure the accuracy of the evaluation, the observer needs to complete the plausibility evaluation within three seconds in each group of comparison experiments.

![Figure 7. Comparison results on evaluation between different edge detection methods.](image)

Figure 7 shows that the plausibility of Canny method is the lowest, and the plausibility of Sobel method is better than Canny method, but less than the plausibility of our method. It proves that our edge detection method in this paper is more suitable for edge detection in texture generation. The comparison of texture generation effects based on different edge detection methods is as follows:

![Figure 8. Comparison between our edge detection method with two classic algorithms.](image)

Figure 8 shows that the texture color based on Canny method is discrete, and the coverage of texture color based on Sobel method is low. However, the method of this paper has a more uniform and complete color effect, indicating the texture generated based on our edge detection method has better quality.
5. Conclusion
We propose an automatic texture generation algorithm for 3D shapes based on cGAN. Our main contribution is to translate the complex 3D model texture production process into a simple automatic model texture generation approach driven by data. By constructing a cGAN-based texture generation pipeline including model sampling, edge detection, texture coloring and texture mapping to achieve the generation of model texture automatically. In addition, we also demonstrate that our edge detection method based on 3D model information can eliminate the influence of environmental factors, which increases the accuracy of edge detection and improves the quality of texture coloring. The experimental results show that the proposed texture generation algorithm can generate high-quality textures. We also validate the universality of the proposed method through different model data. The algorithm of this paper has better performance in low resolution texture generation, but the generated high-resolution texture quality is not ideal. In the future, we plan to further improve the quality of the generated texture in high resolution.

Acknowledgment
This research was supported by Zhejiang Provincial Science and Technology Program in China (2018C01030).

References
[1] Goodfellow I J, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A and Bengio Y 2014 Generative adversarial nets Int. Conf. on Neural Information Proc. Systems
[2] Denton E, Chintala S, Szlam A and Fergus R 2015 Deep generative image models using a laplacian pyramid of adversarial networks Int. Conf. on Neural Information Proc. Systems pp 1486-94
[3] Mirza M and Osindero S 2014 Conditional generative adversarial nets Computer Science 2672-80
[4] Li C and Wand M 2016 Precomputed real-time texture synthesis with markovian generative adversarial networks European Conf. on Computer Vision pp 702-16
[5] Wang T C, Liu M Y, Zhu J Y, Tao A, Kautz J and Catanzaro B 2018 High-resolution image synthesis and semantic manipulation with conditional GANs IEEE Conf. on Computer Vision and Pattern Recognition pp 8798-807
[6] Wang X and Gupta A 2016 Generative image modeling using style and structure adversarial networks European Conf. on Computer Vision pp 318-35
[7] Kittler J 1983 On the accuracy of the Sobel edge detector Image and Vision Computing 1(1) 37-42
[8] Canny J 1987 A computational approach to edge detection Readings in Computer Vision ed Morgan Kaufmann pp 184-203
[9] Shen W, Wang X, Wang Y, Bai X and Zhang Z 2015 DeepContour: a deep convolutional feature learned by positive-sharing loss for contour detection IEEE Computer Vision and Pattern Recognition pp 3982-91
[10] Hwang J J and Liu T L 2015 Pixel-wise deep learning for contour detection Computer Science
[11] Hertzmann A, Jacobs C E, Oliver N, Curless B and Salesin D 2001 Image analogies the 28th annual conf. on Computer graphics and interactive techniques pp 327-40
[12] Isola P, Zhu J Y, Zhou T and Efros A 2017 Image-to-image translation with conditional adversarial networks IEEE conf. on computer vision and pattern recognition pp 1125-34