Gradient-based 3D mesh style transfer using 2D supervision

Abhishek Lalwani
UMass Amherst
alalwani@umass.edu

Himanshu Gupta
UMass Amherst
himgupta@umass.edu

Rushikesh Dudhat
UMass Amherst
rdudhat@umass.edu

Abstract

Conventionally, style transfer from 2D image to 3D object is done by mapping the texture from the image to the 3D surface. However, it is interesting to see if the shape of the object itself changes as per the style of the image. We have used a neural renderer-based approach to transfer style from 2D image to 3D mesh in our work. Here, by style, we mean the changes in the texture and surface. A polygon mesh is a promising candidate for modeling the 3D world behind 2D images based upon 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 3D model to an image, or rendering, involves a discrete operation called rasterization, which prevents back-propagation. Thus, we have used a neural renderer that approximates gradient for rasterization that enables the integration of rendering into neural networks. We have re-implemented the existing neural-renderer based 2D to 3D mesh editing and texture mapping approach using the latest libraries like PyTorch, Cuda, etc.

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.

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 are 3D extensions of pixels, are the most widely used format in machine learning because they can be processed by CNNs [2, 3]. 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, after experimenting with different 3D representation, 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.

Can we train a system including rendering as a neural network? This is a challenging problem which is solved by making the rasterization step differentiable. Hirohu et.al[8] provided once such approach called neural renderer

In this project, we worked on one application of neural renderer presented by the paper Neural 3D Mesh Renderer [5], i.e. gradient-based 3D mesh editing with 2D supervision. This includes the 3D version of style transfer[6]. This task cannot be realized without a differentiable mesh renderer since voxels or point clouds have no smooth surface, and the textures can't be captured in them. The major contributions of this project can be summarized as follows:

- We performed 3D style transfer using voxel, point-cloud, and mesh-based 3D representation and compare the results.
- We performed 3D style transfer using Multiview and single-view object reconstruction and present the findings that reconstruction does not capture the essence of texture as good as 3D mesh deformation.
- We finally present PyTorch implementation and optimization of gradient-based 3D mesh editing pipeline presented in the paper Neural 3D Mesh Renderer.

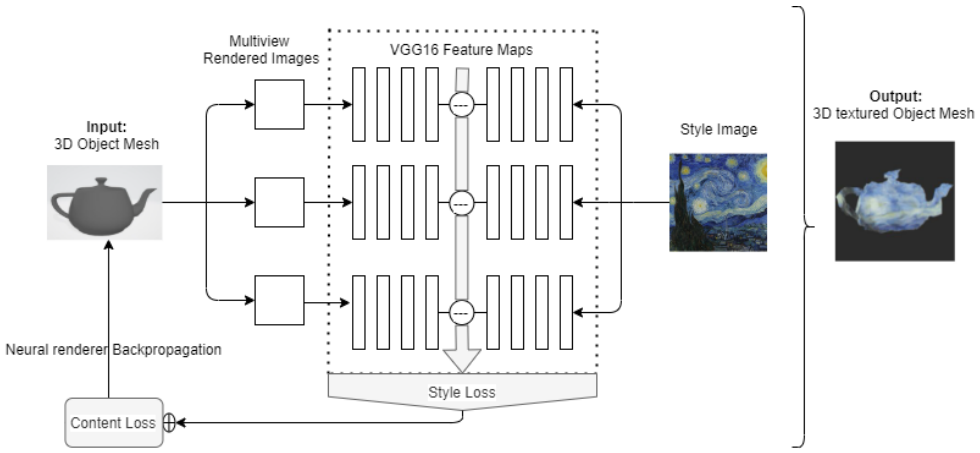

Figure 1. Work flow diagram for the image classification server.

- Gradient-based 3D mesh editing with 2D supervision. This includes a 3D version of style transfer [6] and DeepDream [7]. This task cannot be realized without a differentiable mesh renderer because voxels or point clouds have no smooth surfaces.

2. Related Work

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. The neural renderer [1] by Hirohu et.al provides an approximate way to backpropagate the losses directly to the mesh. This opens a wide range of 3D based applications to configure the meshes.

Previously, the work has been done to map the textures from images to 3D meshes. Using a differentiable feature extractor and loss function, an image that minimizes the loss can be generated via backpropagation and gradient descent. DeepDream [7] 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. 3D mesh editing based on neural rendering was also tried previously by Hiroharu et.al [8]. However, this approach was implemented using Chainer and TensorFlow. This implementation requires a lot of system setup time to work on newer Frameworks and hardware [8]. In this paper, we have re-implementated neural renderer using PyTorch and other latest libraries to provide seamless execution and to provide easy future extensions.

3. Gradient-based 3D mesh editing

Gradient-based image editing techniques [6, 18] generate an image by minimizing a loss function 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 vi, faces fi, and textures ti based on its rendered image $R(m|i)$.

3.1. Style transfer Method and Loss Functions

In this section, we describe the method to transfer the style of an image x s onto a mesh m^c, that is proposed in the neural Renderer paper [8].

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 $L_c(x|x^c) = |f_c(x) - f_c(x^c)|_2^2$. Style loss is defined using another feature extractor $f_s(x)$ and style image x_s as $L_s(x|x_s) = |M(f_s(x)) - M(f_s(xs))|_F^2 *$ $M(x)$ 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 f_i are the same for both meshes, content loss is redefined as $L_c(m|m^c) = Pv_i, v_i^c \in (m, m_c)|v_i v_i^c|_2^2$. Style loss is same as that in the 2D application. Specifically, $L_s(m|x^s, \phi) = |M(f_s(R(m, \phi))) - M(f_s(x^s))|_F^2$. A regularizer for noise reduction is also used alongside with the above mentioned loss. Let P denote the a set of colors of all pairs of adjacent pixels in an image $R(m,)$. We define this loss as $L_t(m|\phi) = \sum_{(p_a,p_b)\in P} |p_a - p_b|_2^2$. The final objective function is $L = \lambda_c L_c + \lambda_s L_s + \lambda_t L_t$. Initial solution

of m is set as m^c and L is minimized with respect to v_i and t_i.

3.2. Architecture

This section describes the complete network architecture of gradient based 3D mesh style transfer that we implemented in our work using PyTorch and Cuda frameworks. A neural renderer architecture is leveraged for this application and a style transfer component is built on top of that. The same is illustrated in the figure 1. A 3D mesh object is fed to the neural renderer pipeline which outputs corresponding multiview images. A VGG16 network is used to extract features from those images which are compared with those extracted from stylized feature image that is also given as an input to the network. We have specifically used features from 2^{nd}, 7^{th}, 14^{th} and 21^{st} layer of VGG network to extract most meaningful textures. Now a style loss is calculated by summing over the differences for each feature map output of rendered and textured image. This loss signifies the texture captured by the mesh. Another loss is calculated which represents the degree by which original mesh has been deformed to capture the texture present in the style image. This is termed as content loss. Both the losses are used to counter balance themselves which leads to stable deformation in the mesh.
Now, a weighted sum of all the losses is taken which is back-propagated to train the network parameters i.e., texture and vertices of 3D mesh object. The deformed mesh is then again fed to the network along with the style image and this whole process is repeated for multiples epochs.
The result of the run is a mesh with deformations and textures that mimics the texture from the given style image. The results of our experiments and work can be seen in the next section.

3.3. Experiments

Our experiments mainly focused on two aspects of style transfer. We tried multiple methods to change the shape of the object according to an input image. Secondly, we also tried multiple methods to transfer texture from a reference image to a 3D object. Although the later was straightforward with a lot of available approaches in literature, the former part required serious experimentation with different approaches.
Initially, we extracted multi-view images from the ground truth mesh. Then we did texture mapping on multi-view images from a reference image and recreated voxelized 3D object from the stylized image. Here we implemented the 3D voxel reconstruction that is proposed by the paper [10]. However, in this case the voxelized approach was not aesthetically appealing and did not capture the texture changes in 2D images.
We then tried recreating 3D mesh using the style transfered single image. We tried this approach using MeshRCNN[9]. In this approach, the recreated mesh was not accurate as well and had large reconstruction error,as shown in figure 2.

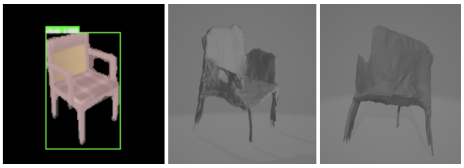

Figure 2. Mesh creation from single styled image

3.4. Experimental setting of the 3D mesh editing network

We re-implemented the existing chainer based 2D to 3D style transfer using the latest frameworks(PyTorch, cuda 11.2) and we could train the mesh to match the style of the reference image. 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 244 × 224. The batch size was set to 4. During optimization, images were rendered at random elevations and azimuth angles. Texture size was set to st = 4. For style transfer, the style images we used were selected from [1]. λc, λs, and λt are manually tuned for each input. The feature extractors for style loss were conv1-2, conv2-3, conv3-3, and conv4-3 from the VGG16 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 learning rate for vertices and texture was set to $2.5e4$ and $5e2$. The pipeline was trained for 100 epochs.

4. Conclusion

In our PyTorch re-implementation of 2D to 3D style transfer we could train the texture and style of the 3D object with the supervision of styled image. These results can be found in the figure 3. We can see that the edges and curves are being captured effectively in the resultant 3D object, according to the style image passed in the input. Although this can be further optimized using many ways of which some are discussed in the next section. Also, the earlier implementation of chainer took 5000 epochs to train the network, which we were able to achieve reasonably in only 100 epochs.

4.1. Future Works

We observed that for every styled image there is a different optimal learning rate. It would be interesting create a pipeline to automatically converge based on some learning rate space. It would be also interesting to use ResNets or other encoders for feature extraction and optimized loss functions like cyclic loss.

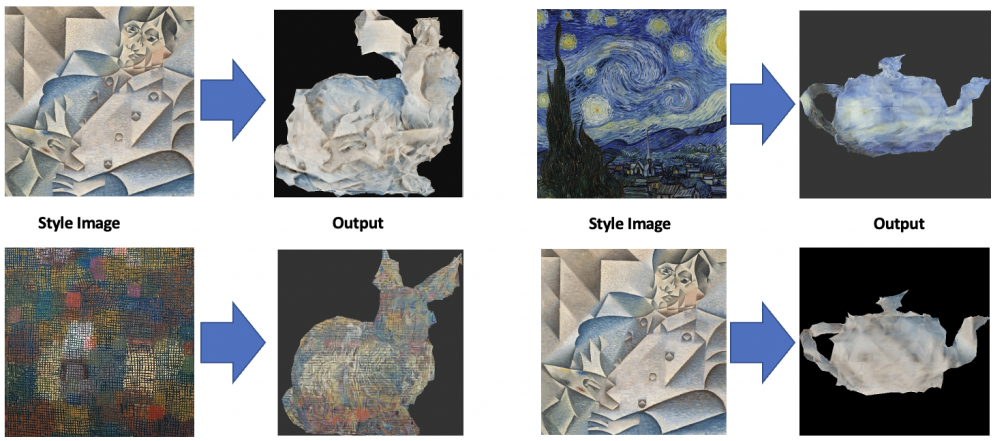

Figure 3. 3D Mesh editing results

5. References

[1] Krizhevsky, Alex Sutskever, I Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. 1097-1105.

[2] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3dr2n2: A unified approach for single and multi-view 3d object reconstruction. In ECCV, 2016.

[3] 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.

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

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

[6] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks.

[7] A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks. Google Research Blog, 2015.

[8] H. Kato, Y. Ushiku, and T. Harada. Neural 3d mesh renderer. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3907–3916, 2018.

[9] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh R-CNN. CoRR, abs/1906.02739, 2019.

[10] Bo Yang, Sen Wang, Andrew Markham, and Niki Trigoni. Robust attentional aggregation of deep feature sets for multiview 3D reconstruction. IJCV, 2019