3D Model Generation from Single-Image View using Adversarial Training and Latent Space Representation

Evaldo BORBA Toshiyuki AMAGASA

Graduate School of Science and Technology, University of Tsukuba 1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8577 Japan
E-mail: evaldo.lborba@kde.cs.tsukuba.ac.jp, amagasa@cs.tsukuba.ac.jp

Abstract The demand for 3D models is increasing steadily in recent years, both in the digital world being used for entertainment purposes thanks to VR and the real world thanks to 3D printing. Therefore, it is important to find a way to create 3D models more easily. Previous researches have already tackled this matter in different ways, relying on databases of 3D models or the use of deep neural networks. However, many of these approaches rely on multiple inputs or the resolution of the generated 3D models are still low. In this research, we developed and proposed a network capable of generating a realistic 3D voxel model by using only one 2D image as an input. In our proposed method, a variational autoencoder neural network (VAE) is introduced to improve the generation of the latent vector and adaptative learning is used to avoid the problem of vanishing gradients. The results obtained from the experiments demonstrate that our method can achieve good results and generate high-resolution 3D models compared to previous researches.

Keyword 3D Model Generation, Generative Adversarial Training, Variational Autoencoder, Unsupervised Learning

1. Introduction

By definition, a 3D model is a mathematical representation of “something” (it can be either a living creature, an inanimate object, or even something that does not exist in real-life) in a three-dimensional world. In recent years, 3D models became more and more common in our everyday life. For example, in the entertainment industry, being often found in movies, advertisements, music videos and more commonly, in games. Since the first big 3D game called Battlezone, back in 1980, 3D models became more and more common for this type of media, to the point that is very hard to find a big game today that does not use any type of 3D asset. It has also been applied for the training of people, being used in simulators for driving, piloting, or even training doctors on how to perform surgeries. It has also been applied for visualization and architecture, transforming ideas and designs into an easy-to-understand result and enabling clients to view and understand the design in an easier way, without incurring any material costs. It has even left the digital world thanks to 3D printing, allowing people to model and print their own house, resulting in lower costs and time saving compared with traditional construction methods [1], it has also reduced the cost of prosthetic limbs, allowing easier access to high-quality prosthetics for people with disabilities [2], and it also has been used to print organs for patients, with ear transplants already being a reality[3], while fully-functional printed hearts are already in development [4].

With 3D models becoming more and more prominent in our daily lives, it is important to understand how they are created. The workflow of how to create one varies from person to person, and from what is being modeled. But independent of who, how or what, it has something in common for all cases, it takes time. The time to create a very realistic and detailed 3D model can vary from a few hours to even days. But thanks to the recent advances of generative models, more specifically Generative Adversarial Networks (GANs) and Convolutional Neural Networks (CNNs), we developed and proposed a network capable of generating a realistic 3D voxel model taking only a few seconds and using only one 2D image as an input.

In our proposed method, a variational autoencoder neural network (VAE) is introduced to improve the generation of the latent vector used as an input for the generative network, and adaptative learning is used to avoid the problem of vanishing gradients [5], where the discriminative network is too good, that it cannot provide enough information for the training of the generative network. The results obtained from the experiments demonstrate that our proposed method can achieve good results and generate high-resolution 3D models.

2. Related Work

2.1. 3D Model Generation

3D model generation is an old problem in computer science and graphics. Since it started decades ago, many

techniques were proposed to solve this type of problem. The most common approach is by using algorithms that synthesize the new models by looking at a 3D database and combining the parts and shapes found or even deform the existing ones to create the desired 3D shape. With [6] being one of the most recent works using a variation of this approach. By using a probabilistic deformation and a generative surface model, it was able to achieve improved shape correspondence. The advantage of this technique is that is able to generate realistic 3D models but requires both a database where it can borrow parts from, together with supervised training.

2.2. Deep Learning for 3D Generation

With the rapid development of deep learning techniques in recent years, many types of deep neural networks were proposed to solve the problem of 3D model generation. [7] attempted to generate 3D models by using Recurrent Neural Networks, taking one or more images from different viewpoints and outputs the reconstruction of the object in the form of a 3D grid. [8] employed an autoencoder network to generate a voxel representation from multiple views. By using Convolutional Deep Belief networks, [9] was able to learn the distribution of 3D shapes from different categories. These approaches have something in common, or they need multiple views from the object that they want to generate, or the final resolution of the 3D object is small. In comparison with these, our approach only needs one view from the object and is able to generate 3D models in a higher resolution compared to the previously mentioned approaches. Our network is able to generate outputs with the resolution of 128x128x128 while [7] is 32x32x32, [8] is 20x20x20 and [9] is 30x30x30.

2.3. Inverse Rendering

Inverse Rendering tries to estimate the physical attributes of a scene (geometry of the 3D model, lighting, maps, materials, …) from multiple views or a single image. Inverse rendering algorithms can be divided into three types: optimization-based, learning-based and differentiable renderer. Optimization-based approaches are able to estimate a few attributes by making assumptions about the statistical priors of the illumination and/or reflectance of images. [10] was able to factorize a scene into a normal map, albedo map and spherical harmonics lighting by using priors similar to those found in natural image statistics, while [11] proposed an optimization method to obtain the reflectance and illumination for indoor scenes by using RGBD sensors and leverage auto-exposure. Learning-based approaches make use of deep learning techniques, mainly Convolutional Neural Networks, to solve this problem. Lighting can be estimated as an environment map, spherical harmonics or point lights [12, 13, 14], depth and normal maps can be estimated with very good results [15], and more recently, [16] proposed a new method capable of jointly estimate the albedo, normal and lighting of an indoor scene. Differentiable rendering algorithms try to estimate the partial derivatives of pixels in a rendered image with respect to some scene parameters. The first differentiable renderers [17] were able to compute the gradients with respect to simple material parameters but failed to handle more complex ones like the vertex positions of a model and the camera. More recent works using differentiable rendering, like [18], can already reconstruct the 3D geometry and materials of real-world objects from reference photographs and an initial approximation by using standard Monte Carlo sampling with automatic differentiation.

While these approaches can already achieve good results at the estimation of the scene and 3D model parameters, they tend to not focus much on the 3D model itself.

2.4. Generative Models

Since it was introduced by [19], Generative Adversarial Networks (GANs) has attracted a lot of attention for the generative field. In image synthesis, [20] was able to achieve impressive results, being able to generate very realistic high-resolution images (1024x1024 pixels) of different types of objects and human faces. [21] has successfully used GANs for image inpainting and restoration, by generating novel fragments that were not present elsewhere in the image. [22] used GANs to create a network capable of drawing realistic pictures by using image labels to control the contents of the image and a reference style image to control the appearance. Meanwhile, [23] proposed a two-stage GAN to generate images from a text description through a sketch-refinement process. Like the approaches mentioned above, many GANs focus on solving problems related to 2D images, in this research, like [24, 25, 26], we focus on translate this adversarial approach for the generation of 3D objects. Some differences between this research and [24, 25, 26] include the layout of the networks and also the resolution of the final 3D models, 128x128x128 against 64x64x64 used for both [24,25] and 32x32x32 for [26].

3. Approach

In this section, we introduce our implementation for the generation of 3D models. We start with the dataset used for the training of the network, the architecture, and how to train all the networks.

3.1. Data Representation

While 2D images have a common way to be represented in computers (by using the values of the pixel), 3D data can be represented by many formats. Polygonal mesh is the most common way to represent a 3D model in most of the 3D modeling software, meanwhile, point cloud representation is often used for CAD related applications, and more recently, voxel grids also became an option. A voxel (short for volumetric pixel) is the three-dimensional equivalent of a 2D pixel. Like the 2D counterpart, voxels do not contain the information about the location in a 3D space explicitly on their values. These coordinates are inferred based on the position relative to the nearby voxels. While pixels data represents the color information, voxels data generally indicates the existence (or not) of a "cube" at that point in space. When put together, these cubes can represent the geometry of a 3D model, just like Lego pieces in the real world. Even though cubes are the most common shape for representation, voxels can be of any shape if they can fulfill two conditions, the shape has to be the same throughout the model and when placed next to each other it should give a gap-less impression. In contrast to point clouds, voxels can only inhabit discrete positions in the grid and all possible positions are equally spaced. This representation is useful because it can be directly applied to neural networks like CNNs. On the other hand, this representation is rather sparse for high-resolution 3D models. Because voxel grid requires less memory and can be directly applied to CNNs, we use this type of representation for the experiments in this paper. The conversion from a .obj and .off files to a voxel grid is done thanks to [27, 28].

3.2. Training Dataset

For the training of the network, we need a 2D image and the corresponding 3D model associated with it. We take the 3D models from the public ShapeNet dataset [29]. The ShapeNet dataset is composed of 51.300 unique 3D models divided into 55 categories. For the experiments part, we have used only five of these categories so far, airplane (4.045 models), chair (6.775 models), car (3.514 models), sofa (3.173 models) and table (8.433 models). For the generation of the 2D image, each model is rendered by using Blender [30] in 30 different views, by rotating the model 12 degrees along the y-axis, with slight modifications to the default camera and illumination parameters. At first, we tried to combine the obtained render with a background image, trying to simulate a real-life scenario, where the 3D model that we want to generate is based on a picture of the object itself. While airplane renders could be easily combined with sky images, other types of objects proved to be quite difficult to find a good match. For example, the combination between chair renders and living room images. The scale of the chair has to be similar to objects already in the picture and the position cannot overlap with another furniture present in the image. Even if we could find a solution for these problems, the lighting for the rendering is also very different from the natural lighting found in real-world pictures, often resulting in high contrast between our 3D model and the background picture. In the end, we decided to use the render with a transparent background as an input for the networks.

Fig 1: Render results of each class

3.3. Architecture

Generative Adversarial Networks (GANs) proposed by [19], consists of two different neural networks, a discriminator and a generator. The generator network is responsible for generating something (this something can be images [20,21,22], audio [31], or in our case, 3D voxel models) using a latent vector z as an input. On the other hand, the discriminator network is responsible to classify between real objects from a dataset and objects created by the generator. In our approach, the Generator G maps a 200-dimensional latent vector z to a 128 x 128 x 128 voxel model, which represents the object in voxel space. We call this object $G(z)$. The discriminator D outputs a value

between 1 and 0, where 1 means that the input comes from a real data distribution (from a dataset), and 0 means that the object is a "fake one" (comes from the generator). We represent both of these values by *D(x)* and *D(G(z))* respectively, where *x* is an object from the ShapeNet dataset. To create the latent vector *z*, most approaches using GANs, sample it from a random probabilistic latent space, but inspired by [32], we infer this latent vector *z* from observations. To do that, we use an additional network, a Variational Autoencoder (VAE) [33], which is composed by an encoder *E*, which maps a 2D image *y* to the parameters of a statistical distribution (mean and variance) which we use to sample our latent vector *z*, and a decoder *Dec*, which maps the latent vector *z* back to its original image. The effect of the variational autoencoder network is demonstrated in the experiments section.

The details of each network are as follows.

Generator: The input is composed of a 200-dimensional vector and outputs a 128 x 128 x 128 matrix with all values between 0 and 1. The generator is composed by six deconvolution layers with number of channels equals to [512, 256, 128, 64, 32, 1], kernel sizes equals to [4, 4, 4, 4, 4, 4], and strides [1, 2, 2, 2, 2, 2]. We add a batch normalization layer after the first five deconvolution operations and a ReLu activation layer for each one, and a Sigmoid layer at the end of the network.

Discriminator: The input is a 128 x 128 x 128 matrix and outputs one real value between 0 and 1. The discriminator, like the generator, is also composed of six convolutional layers, with the number of channels equals to [32, 64, 128, 256, 512, 1], kernel sizes [4, 4, 4, 4, 4, 4] and strides [2, 2, 2, 2, 2, 1]. Also have batch normalization layer after the first five convolutions and a leaky ReLu of value 0.2 after each one, with a Sigmoid layer after the last convolutional layer.

Encoder: The encoder receives a 256 x 256 x 3 image as an input, and outputs two 200-dimensional vectors, one representing the mean and the other the variance. It is composed of six convolutional layers followed by two linear layers at the end. The size of each convolutional channel is equal to [32, 64, 128, 256, 512, 1024], kernel size [4, 4, 4, 4, 4, 4], and strides [2, 2, 2, 2, 2, 2]. The output of the last convolutional layer goes through a linear layer of size 1024 which is connected to two layers of dimension 200. After all the convolutional layers there is a batch normalization layer, and after all layers, a leaky ReLu activation.

Decoder: The decoder receives a 200-dimensional vector as an input and outputs an image of the dimensions 256 x 256 x 3. It is composed of two linear layers followed by six convolutional layers. The size of linear layers are 1024 and 16384 (4x4x1024) respectively, and the size of each convolutional channel is equal to [512, 256, 128, 64, 32, 3], kernel size [4, 4, 4, 4, 4, 4] and strides [2, 2, 2, 2, 2, 2]. After all the convolutional layers there is a batch normalization layer, and all layers are followed by a leaky ReLu activation layer of value 0.2.

Figures 2 and 3 show how the architectures of the networks are organized.

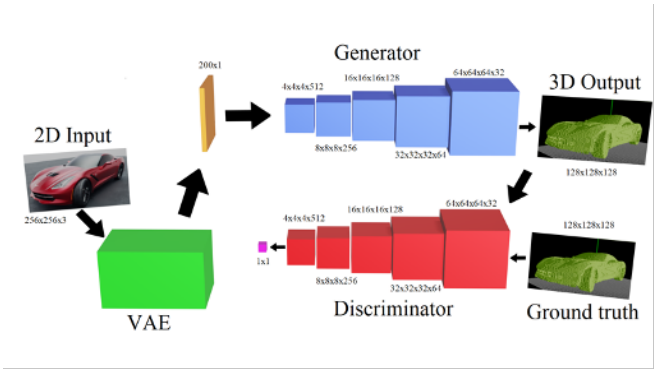

Fig 2: The architecture of all networks

Fig 3: The architecture of the VAE network

3.4. Training

First, we train our six VAE networks (one for each object type and one for all the objects) how to map the input images to a latent vector *z*. We divide the dataset of each object into a training, validation and test dataset. Because the datasets have different sizes, we divide them in different ways. For the airplanes, cars and sofas dataset we separate 500 models for validation and 500 for test. For the chairs dataset, we use 750 for validation and 750 for test. While for the tables dataset we separate 1000 different models for validation and test. For the all objects network, we kept 500 different objects of each type for validation and test. All the remaining models were used for the training.

The parameters of our VAE network are trained by using two losses, a reconstruction loss L_{recon} and a regularization loss L_{reg}. The L_{recon} is used to force the decoded image to match the original input, while the L_{reg} is used to improve

the latent space and reduce the overfitting to the training data.

$$L_{recon} = -\frac{1}{N}\sum_{i=0}^{n}[z_i\, log(\check{z}_i) + (1 - z_i)\, log(1 - \check{z}_i)], \quad (1)$$

$$L_{reg} = -5e^{-4} * \mu(1 + z_{var} - {z_{mean}}^2 - e^{z_{var}}), \quad (2)$$

$$L_{VAE} = \mu(L_{recon} + L_{reg}), \quad (3)$$

Where N represents the number of samples, z is the ground truth and $\check{z}$ is the predicted output, and z_{mean} and z_{var} are the mean and variance output of the encoder network.

For the optimization of the VAE network we use ADAM [34] with $\beta 1 = 0.9$, $\beta 2 = 0.999$, learning rate equals to 0.001 and decay of 1e-8.

The usual procedure for training a GAN is to update the weights of both generator and discriminator at every iteration. However, because the generation of objects in a 3D voxel space is a more complicated task than classify between real and fake objects, the generator network usually learns slower than the discriminator. This causes the discriminator to classify the fake objects with very high confidence early on during the training, making it harder for the generator to obtain a good error value to modify its own weights. To solve this problem, we employ "adaptive learning". For each iteration, we only update the weights of the discriminator, if and only if, the accuracy was lower than 70%, this ensures that both networks learn at almost the same pace and stabilize the training, producing more realistic results. The effect of adaptive learning during training is demonstrated in the experiments section.

The loss function of our network is divided into two parts, L_D and L_G. L_D and L_G are the cross-entropy loss for the Discriminator and Generator Network respectively, while L represents the total loss of the network.

$$L_D = log(D(x)) + log(1 - D(G(E(y)))), \quad (4)$$

$$L_G = log(D(G(E(y)))), \quad (5)$$

$$L_{GAN} = L_D + L_G \quad (6)$$

For the optimization we use ADAM [34], with $\beta 1 = 0.5$ and $\beta 2 = 0.999$. And learning rate of 0.0025 and 0.00001 for the generator and discriminator networks respectively.

4. Experiments

To evaluate our approach, we performed some experiments. We first perform the training of our VAE networks, to see how well it can learn the mapping from image to latent vector, after that we show qualitative and quantitative results of the objects generated by our GAN, and we also performed two ablation studies, one to show the advantage of the Variational Autoencoder network and another one to show how much the adaptive learning parameter in the discriminator network affects the output of the generator network.

4.1. VAE Result

Table 1 and figure 4 shows the average loss of each network on the test dataset at the end of the training. We trained each network for 250 epochs, using batch size equals to 16. For each iteration, we take random images from the training dataset and update the weights of the network according to equation (3).

As we can see from the results, all the networks were able to achieve a similar loss value at the test dataset, with the exception of the airplane network, which had a lower loss compared to the others. We believe that the reason for that is because of the shape of airplane objects. As we can see from figure 1, they look smaller in the images compared to the other types. Consequently, causing the "area of interest" inside of the image to be smaller, making it easier for the network to learn the representation and causing a lower loss value compared to the others.

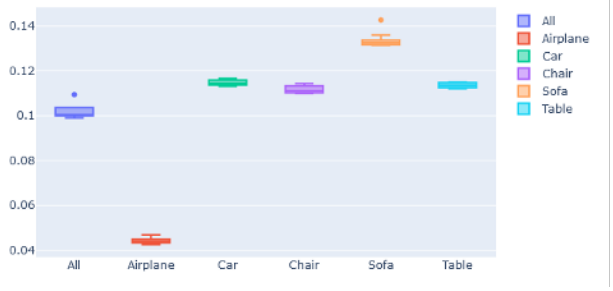

Fig 4: VAE loss on the test dataset

Table 1: Loss error for the VAE network

Network	Loss Value
All	0.1021 ± 0.0037
Airplane	0.0443 ± 0.0012
Car	0.1147 ± 0.0011
Chair	0.1117 ± 0.0017
Sofa	0.1338 ± 0.0032
Table	0.1136 ± 0.0010

4.2. Qualitative Results

Figures 5 to 10 shows some of the 3D objects generated by our networks along with the respective image used as an input. For this experiment, we trained one network for each of the five objects category, plus one more for all objects. We train each network for 75 epochs, using a batch size equal to 8. For each iteration, we take a pair $\{x_i, y_i\}$, where x_i is a 3D object from the dataset and y_i is a random render from that object. From y_i, we generate a

latent vector z_i using the corresponding VAE network, and the 3D object $G(z_i)$ using the Generator network. Finally, we run the discriminator network, using the x_i and $G(z_i)$ and update the networks using their respective loss function (Equations (4) and (5)).

Even though it is harder to generate high-resolution objects, due to the rapid growth of 3D space, our networks are able to achieve good results for airplane, car and sofa objects. On the other hand, the network for all, chair and table were not able to get the desirable results, often resulting in objects with some differences in shape or problems with continuity. We assume that the difference in quality between the results is because of the implemented training configuration. All six networks were trained using the same hyperparameters (learning rate, weight decay, optimizer, …) and network architecture. To solve this problem and improve generation, we plan in the future to fine-tune each network to find the best parameters/architecture that can achieve the best result for each object.

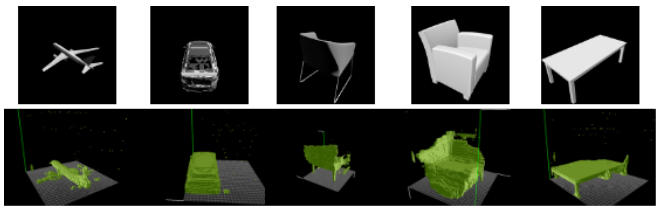

Fig 5: Results from the all dataset

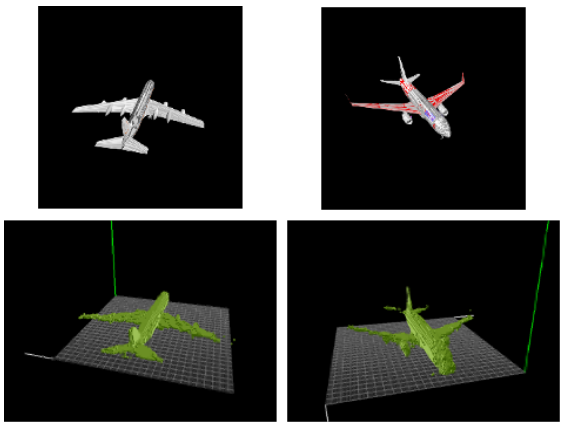

Fig 6: Results from the airplane dataset

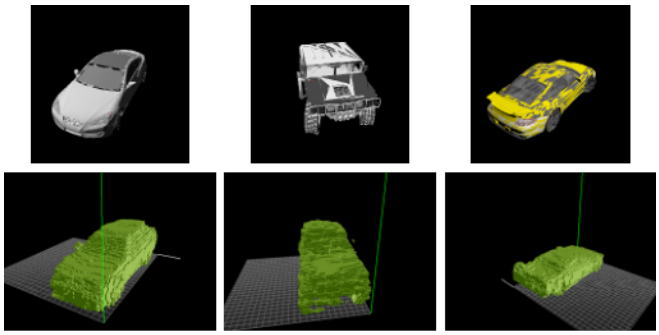

Fig 7: Results from the car dataset

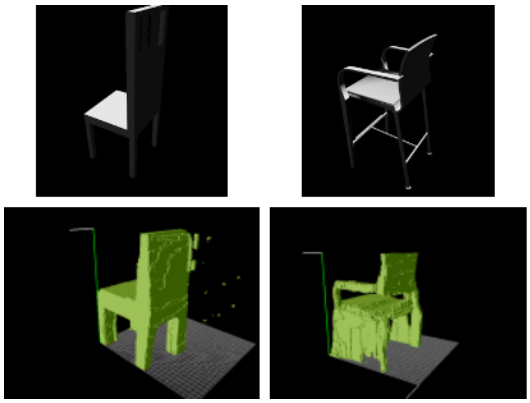

Fig 8: Results from the chair dataset

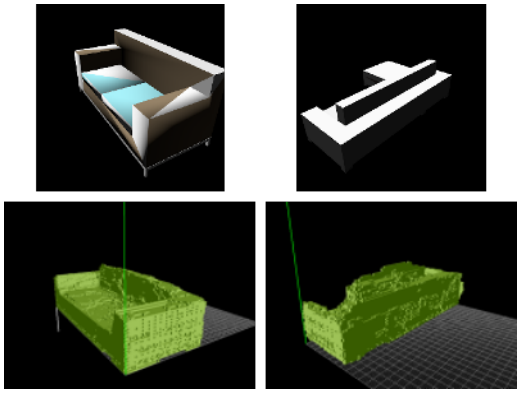

Fig 9: Results from the sofa dataset

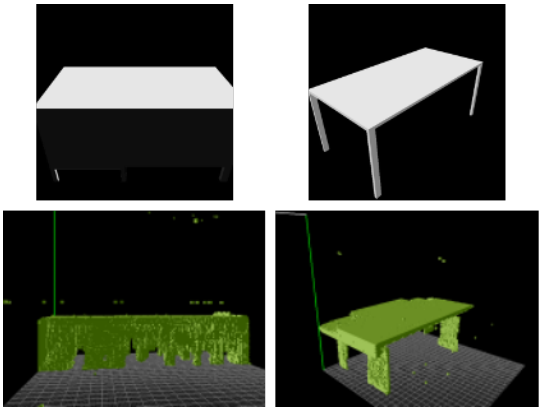

Fig 10: Results from the table dataset

4.3. Quantitative Results

To evaluate in a quantitative way the models generated by our proposal, we employ a feature extraction in our discriminator network. The idea here is to see how well the network learns the features of 3D models by using them as an input for a classification problem. We do that by replacing the last layer of our network with a class label layer. We fine-tune this network using the ModelNet dataset created by [9, 35], more specifically the ModelNet10 and ModelNet40 datasets. ModelNet10 and ModelNet40 are datasets composed of CAD models from 10 and 40 different types of objects respectively. Here we follow the same train/test split used by [9], where approximately 80% of the models are used for training (3991 and 9843 models) and the remaining 20% (908 and 2468) are used for testing. For this dataset, we apply the same render procedures as the ShapeNet dataset. We fine-tune during 100 epochs and use the per-category accuracy to evaluate the performance. The results of our comparison against [24, 25, 26, 39] can be seen in the table below.

Classification			
Method	**Input type**	**ModelNet10**	**ModelNet40**
[9]	3D(30x30x30)	83.5%	77.3%
[36]	3D(28x28x28)	93.8%	-----
[37]	2D Images	94.71%	93.4%
[38]	2D Images	**98.46%**	**97.37%**
Generation			
Method	**Resolution**	**ModelNet10**	**ModelNet40**
[25]	64x64x64	91%	83.3%
[24]	64x64x64	91.63%	**87.85%**
[26]	32x32x32	92.2%	86.4%
[39]	32x32x32	**92.4%**	-----
Ours	128x128x128	90.05%	83.1%

Table 2: ModelNet accuracy

The first four rows summarize the accuracy of the networks trained focusing on the classification of 3D objects. Meanwhile, the bottom five, are networks trained for the generation. The generative networks have shown results not that far behind the ones trained for classification, showing that they were able to learn features of 3D models, even though they were not trained for that explicitly. Between the generative models, [39] and [24] were the ones with the best accuracy on the ModelNet10 and ModelNet40 respectively. Our approach was able to achieve 90.05% and 83.1% on the ModelNet10 and ModelNet40 respectively, lagging a few points behind [39] and [24], the ones with the best accuracy between the generative models. But one important thing to notice is the output resolution of the generated 3D objects, our proposed approach is the only one able to generate 128x128x128 models, which is much harder to obtain due to the rapid growth of 3D space.

4.4. Ablation Study

For the ablation study, we retrained the airplane network a few times, to judge the contribution of the VAE network and adaptive learning on the final output. For the first ablation study, we trained three networks: one without any type of encoder, pulling the input vector straight from a random distribution, one using an encoder network which is trained together with the generator network [32] to map the image to a latent vector and our approach using the fully trained VAE network. For the second one, we trained two networks, one with and the other without the adaptive learning, where both generator and discriminator are updated on every iteration.

Some results from the ablation study can be seen in figure 11. As can be noticed from the models obtained by the network without an encoder, the results are not realistic enough and, on top of that, some noise can be found all around the 3D objects. By using a simple encoder only, the results were able to improve considerably, but some problems can still be noticed, like continuity (horizontal stabilizer on the top airplane and the turbine on the bottom one) and some noise around the front part of the object.

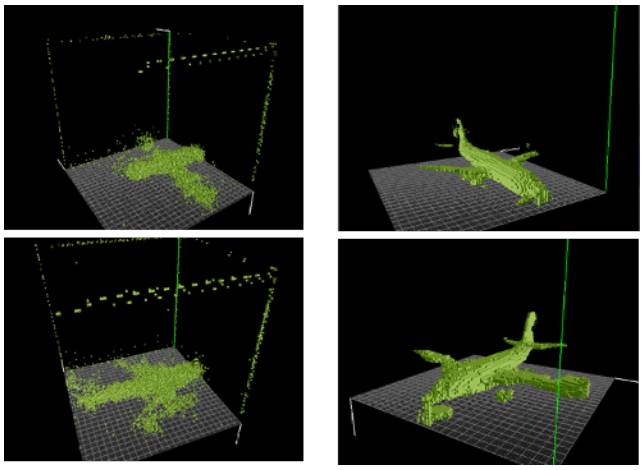

Fig 11: Ablation study. left: no encoder, right: normal encoder

We can observe similar results for the network which does not use adaptive learning. In this case, not using adaptive learning causes the L_D to always be close to zero at the beginning or halfway through the training, while the L_G fluctuates between a low and high loss value. As a result, the discriminator network ends up learning way faster than the generator network, causing it to stop learning useful information, getting the results that can be seen from figure 12.

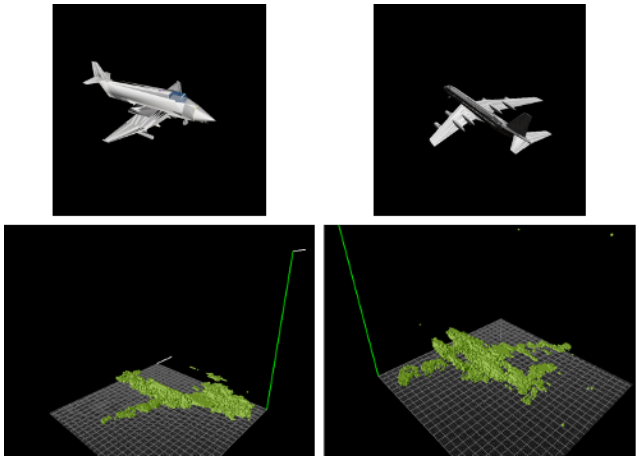

Fig 12: Ablation study, without adaptive learning

5. Conclusion and Future Work

In this research, we trained a network for the generation of 3D models using adversarial training. We demonstrated that with only one 2D image we are able to achieve realistic results on some types of objects. We also showed the importance of the inclusion of the Variational Autoencoder network and the advantage of using adaptive learning on the discriminator to generate better outputs. And also proved that our network, without supervision, can learn the features of 3D objects, achieving a respectable performance on object classification. But some improvements can still be made, ranging from the quality of the generated models to the variety of objects that can be generated.

References

[1] I. Hager, A. Golonka, R. Putanowicz, "3D Printing of Buildings and Building Components as the Future of Sustainable Construction?", International Conference on Ecology and new Building materials and products (ICEBMP), 2016.

[2] Alliance of Advanced BioMedical Engineering, "3-D printing to Lower Prosthetics Costs", Retrieved from https://aabme.asme.org/posts/3-d-printing-to-lower-prosthetic-costs.

[3] G. Zhou, H. Jiang, Z. Yin, Y. Liu, Q. Zhang, C. Zhang, B. Pan, J. Zhou, X. Zhou, H. Sun, D. Li, A. He, Z. Zhang, W. Zhang, W. Liu, Y. Cao, "In vitro Regeneration of Patient-specific Ear-shaped Cartilage and Its First Clinical Application for Auricular Reconstruction", EBioMedicine, Vol. 28, 2018.

[4] N. Noor, A. Shapira, R. Edri, I. Gal, L. Wertheim, T. Dvir, "3D Printing of Personalized Thick and Perfusable Cardiac Patches and Hearts", Advance Science, Vol. 6, No. 11, June 2019.

[5] M. Arjovsky, L. Bottou, "Towards Principled Methods for Training Generative Adversarial Networks", ICLR, 2017.

[6] H. Huang, E. Kalogerakis, B. Marlin, "Analysis and Synthesis of 3D Shape Families via Deep-Learned Generative Models of Surfaces", SGP, Vol. 34, 2015.

[7] C. B. Choy, D. Xu, J. Gwak, K. Chen, S. Savarese, "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction", ECCV, 2016.

[8] R. Girdhar, D. F. Fouhey, M. Rodriguez, A. Gupta, "Learning a Predictable and Generative Vector Representation for Objects", ECCV, 2016.

[9] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, J. Xiao, "3D ShapeNets: A Deep Representation for Volumetric Shapes", CVPR, 2015.

[10] J. T. Barron, J. Malik, "Shape, Illumination, and Reflectance from Shading", TPAMI, 2015.

[11] E. Zhang, M. F. Cohen, B. Curless, "Emptying, Refurnishing, and Relighting Indoor Spaces", TOG, 2016.

[12] M. Gardner, K. Sunkavalli, E. Yumer, X. Shen, E. Gambaretto, C. Gagne, J. Lalonde, "Learning to Predict Indoor Illumination from a Single Image", TOG, 2017.

[13] H. Zhou, J. Sun, Y. Yacoob, D. Jacobs, "Label Denoising Adversarial Network (LDAN) for Inverse Lighting of Face Images", CVPR, 2017.

[14] E. Zhang, M. F. Cohen, B. Curless, "Discovering Point Lights with Intensity Distance Fields", CVPR, 2018.

[15] H. Fu, M. Gong, C. Wang, K. Batmanghelich, D. Tao, "Deep Ordinal Regression Network for Monocular Depth Estimation", CVPR, 2018.

[16] S. Sengupta, J. Gu, K. Kim, G. Liu, D. W. Jacobs, J. Kautz, "Neural Inverse Rendering of an Indoor Scene from a Single Image", ICCV, 2019.

[17] G. Patow, X. Pueyo, "A Survey of Inverse Rendering Problems", CGF, 2003.

[18] G. Loubet, N. Holzschuch, W. Jakob, "Reparameterizing Discontinuous Integrands for Differentiable Rendering", SIGGRAPH, 2019.

[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, "Generative Adversarial Nets", NIPS, 2014.

[20] T. Karras, T. Aila, S. Laine, J. Lehtinen, "Progressive Growing of GANs for Improved Quality, Stability, and Variation", ICLR, 2018.

[21] S. Iizuka, E. Simo-Serra, H. Ishikawa, "Globally and Locally Consistent Image Completion", SIGGRAPH, 2017.

[22] T. Park, M. Liu, T. Wang, J. Zhu, "Semantic Image Synthesis with Spatially-Adaptive Normalization", CVPR, 2019.

[23] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, D. Metaxas, "StackGan++: Realistic Image Synthesis with Stacked Generative Adversarial Networks", TPAMI, 2018.

[24] J. Zhu, J. Xie, Y. Fang, "Learning Adversarial 3D Model Generation with 2D Image Enhancer", AAAI, 2018.

[25] J. Wu, C. Zhang, T. Xue, W. T. Freeman, J. B. Tenenbaum, "Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling", NIPS, 2016.

[26] S. H. Khan, Y. Guo, M. Hayat, N. Barnes, "Unsupervised Primitive Discovery for Improved 3D Generative Modeling", CVPR, 2019.

[27] 3D Mesh Voxelizer, "Binvox", Retrieved from https://www.patrickmin.com/binvox.

[28] F. S. Nooruddin, G. Turk, "Simplification and Repair of Polygonal Models Using Volumetric Techniques", IEEE TVCG, Vol. 9, No. 2, pp. 191-205, 2003.

[29] ShapeNet 3D Model Dataset, Retrieved from https://www.shapenet.org.

[30] Blender 3D Software, Retrieved from https://www.blender.org.

[31] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, A. Roberts, "GANSynth: Adversarial Neural Audio Synthesis", ICLR, 2019.

[32] A. B. L. Larsen, S. K. Sonderby, H. Larochelle, O. Winther, "Autoencoding beyond pixels using a learned similarity metric", ICML, 2016.

[33] D. P. Kingma, M. Welling, "Auto-Encoding Variational Bayes", ICLR, 2014.

[34] D. P. Kingma, J. Ba, "Adam: A Method for Stochastic Optimization", ICLR, 2015.

[35] ModelNet 3D Dataset, Retrieved from https://modelnet.cs.princeton.edu.

[36] N. Sedaghat, M. Zolfaghari, E. Amiri, T. Brox, "Orientation-boosted Voxel Nets for 3D Object Recognition", BMVC, 2017.

[37] Z. Han, H. Lu, Z. Liu, C. Vong, Y. Liu, M. Zwicker, J. Han, P. Chen, "3D2SeqViews: Aggregating Sequential Views for 3D Global Feature Learning by CNN with Hierarchical Attention Aggregation", IEEE TIP, 2019.

[38] A. Kazezaki, Y. Matsushita, Y. Nishida, "RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints", CVPR, 2018.

[39] J. Xie, Z. Zheng, R. Gao, W. Wang, S. Zhu, Y. N. Wu, "Learning Descriptor Networks for 3D Shape Synthesis and Analysis", CVPR, 2018.