NERVISION - TEXT TO 3D-MODEL AI

Alex Steiner
Artificial Intelligence Developer
H-Farm International School
Graz, Austria
https://www.alexsteiner.dev
alex.steiner@student.h-is.com

ABSTRACT

Nervision implements a GAN with many changes. Instead of using a random noise vector as input to the generator, a phrase embedding vector with random noise added is used. The generator creates a 2048x3 array, where each value represents the x, y, and z position of a point in 3D space. The array is then up-sampled to obtain a better-resolution point cloud from which to construct a mesh. The process concludes with the application of a texture to the model done in collaboration with Polyhive AI.

***K*eywords** Point Cloud GAN · Text-To-3D-Model · AI

1 Introduction

Artificial Intelligence throughout the last years made a ton of progress becoming more sophisticated and accurate. The first text to image model was developed in 2021, called DALL-E [8], created by the research team of Open AI. It was one of the first times when an artificial intelligence was able to generate photo realistic images, while in 2022 Chat GPT was released, therefore I tried to implement the Generative Adversarial Network (GAN), introduced in 2014 by Ian Goodfellow [5], in a 3D environment.

As mentioned in the original paper, in a GAN there are two separate neural networks: a discriminator and a generator. The job of the discriminator is to attempt to distinguish real and fake data samples, while on the other hand, the generator generates new data samples that are similar, but different, to the training data. During the training process, the two models are competing against each other, with the generator network trying to create samples as realistic as possible with the intend of fooling the discriminator, which has to accurately tell if the generation is realistic or not. [5].

In recent years, interest in the application of generative models to 3D modeling has increased. However, creating 3D models is far more difficult and time consuming than creating 2D images, as both surface (mesh) and texture must be generated in addition to the geometry of the object.

Point-E, developed by the Open AI team, is a new technique for 3D modeling that employs generative models; however, it varies from Nervision, in which its 3D model generation process uses one or more input photos. Point-E creates the object's point cloud using a GAN, which is subsequently transformed into a mesh through a mesh reconstruction technique. [7].

On the other hand, Nervision, creates a 3D model from a simple text description, it can be considered as another approach of 3D modeling based on a generative model. Nervision, as mentioned before, uses a GAN, to create a point cloud of the object, then a surface reconstruction algorithms is used to convert the point cloud into a 3D mesh using Open 3D [11]. The subsequent cross-section is then finalized using a high-goal surface to create a reasonable 3D model. Working together with Polyhive AI [1], then we generate a texture from a random seed which then is smoothly applied to the mesh, creating an unique model generated completely by an artificial intelligence only from a text.

These novel approaches to 3D modeling have the potential to significantly reduce the time and cost of traditional 3D modeling methods while simplifying the creation of complicated 3D models. If the innovation continues growing, it

could become increasingly refined and used all the more widely in various businesses that rely on 3D representations, such as film, design, layout, and game development.

Nervision can be considered an innovative 3D GAN, as it differs from each one presented so far. Both the Warping GAN [10] and the Tree GAN [9] as they are able to generate a model only from a random noise vector, however if wanted to create it from a text this implemented method wouldn't work. Exactly this makes Nervision innovative and different from other already implemented GANs or diffusion models. This is done by taking a sentence embedding (a way to represent text in a vector) adding a random noise to get a different generation each time.

To accomplish this I need a training set, which was kindly provided by Shape Net, while thanks to some friends, together we wrote a description of each model. [4]

2 Related Work

Tree GAN it is one of the GANs where Nervision is inspired the most, as the discriminator is similar the one implemented in the paper, however the generator is completely different since it doesn't have the TreeGCN layers but only 1D convolutions with a LeakyRelu activation. I decided to do so because after doing numerous tests, it turned out to be just over complicating without giving any better results as the quality wasn't improving at all. [9]

Warping GAN as you will find in the proposed method in this paper, our loss implements the improved WGAN training, which to have more stability adds to the discriminator error a gradient penalty. [6]

Wasserstein GAN-GP Nervision is inspired on this GAN because of the code enhancement layer that it implements and the 1D convolutions is the generator. Exactly as shown in the paper, an embedding is augmented and transformed into a geometric sequence, in this case 512. The main difference however is that the Warping GAN is all around based on a grid building method plus a stitching loss, on the other hand Nervision doesn't need it at all. [10]

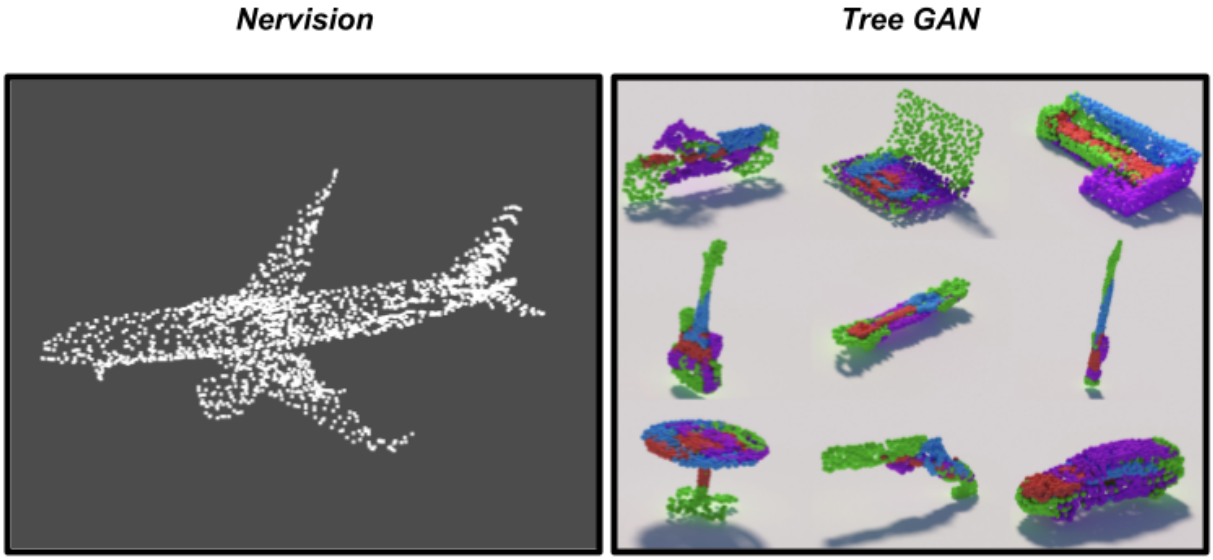


Figure 1: Nervision vs Tree GAN

3 Proposed Method

As mentioned before, Nervision is a GAN. Exactly how mentioned on the original paper written by Goodfellow, there are two different models: a generator and a discriminator. The objective of the generator is to create new data similar to the training data from a vector, while the discriminator is responsible for detecting whether a given data pattern is genuine (from the training set) or false (generated by the generator), giving as output one real number. [5].

The GAN is used in the Nervision implementation to generate 3D models that are comparable to a training set. This is accomplished by feeding the generator a sentence embedding, which is a vector representation of a textual description

of the 3D model being created. Sentence embedding is a natural language processing technique that converts sentences into vectors of real numbers. A random noise vector of the same size as the sentence embedding is added to the input to add some unpredictability to the generation process.

Taking a step back to the sentence embedding, before taking the text and passing it through the universal sentence encoder [3] some natural language processing is performed.

After the input is prepared, it is sent through a few convolutional layers with a leaky ReLU activation function in 512, 1024, 2048, and 2048x3 sizes. The output is converted into a 1x6144x1 slab array, which is reconfigured into a 2048x3 array. Each value in the 2048x3 array corresponds to a point cloud with x, y and z coordinates.

The point cloud is then sent to the discriminator, which determines whether the model created is correct or incorrect. The discriminator is a neural network that has been trained to distinguish between true models from the training set and models generated by the generator. If the discriminator determines that the created model is not close enough to the training set, it reports a certain loss in order to change its parameters and create a better model.

This repeated process of creating models and having them evaluated by the discriminator continues until the generator produces models that the discriminator believes are indistinguishable from the training set. The ultimate goal of the GAN architecture is to create new data patterns that are so similar to the training data that they cannot be confused with the actual examples.

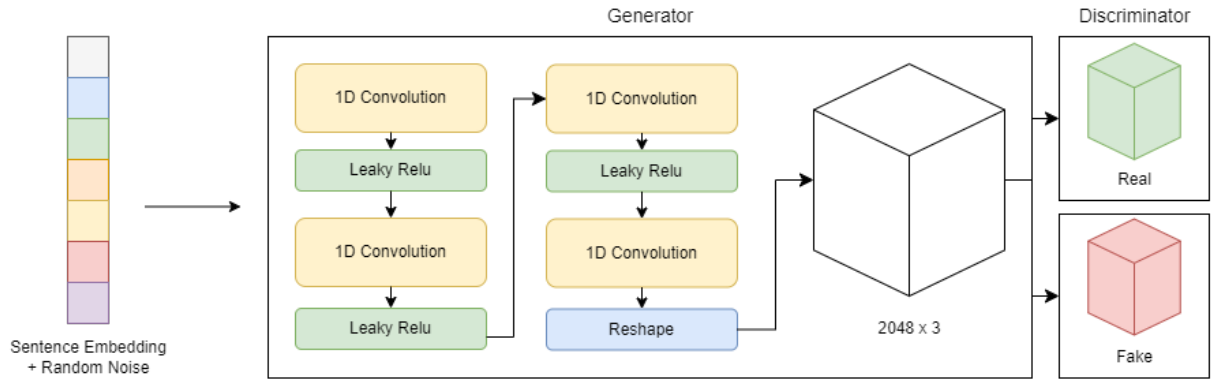


Figure 2: General GAN outline

In the implementation of Nervision GAN, the generator is divided into two parts. The first part takes the sentence embedding as input and runs it through a series of convolutional layers. The second phase of the generator takes the point cloud as input and refines the resulting model by running it through additional layers to build a 2048x3 point cloud.

The sentence embedding is processed by a 1D convolutional layer with a kernel size of 1 and no distortion in the first half of the generator. The activation function of the convolution layer is a LeakyReLU with a slope of 0.2, which introduces non linearity into the model. The activation function for the LeakyReLU is given by max(0.2x, x), where x is the input.

After the first convolutional layer, the output is passed through numerous other convolutional layers, each of which has a LeakyReLU activation function. The philtre sizes of the convolutional layers are 512, 1024, 2048, and 2048x3, respectively. These layers are to convert the input into a flat array of 6144 elements.

The output of the convolution layer is then converted into a 2048x3 array giving to use the point cloud of the generated model. Each 2048x3 item in the array represents a 3D point with x, y, and z coordinates.

Overall, the generator of Nervision's implementation GAN takes a phrase embedding as input and runs it through a series of convolutional layers with LeakyReLU activation functions before refining it through additional layers to create a 2048x3 point cloud. The actual architecture of the generator may differ depending on implementation and training data.

The discriminator consists in a total of ten layers, divided into two halves. The first part consists in convolutions, while the second part consists in a MaxPool1D layer followed by linear layers.

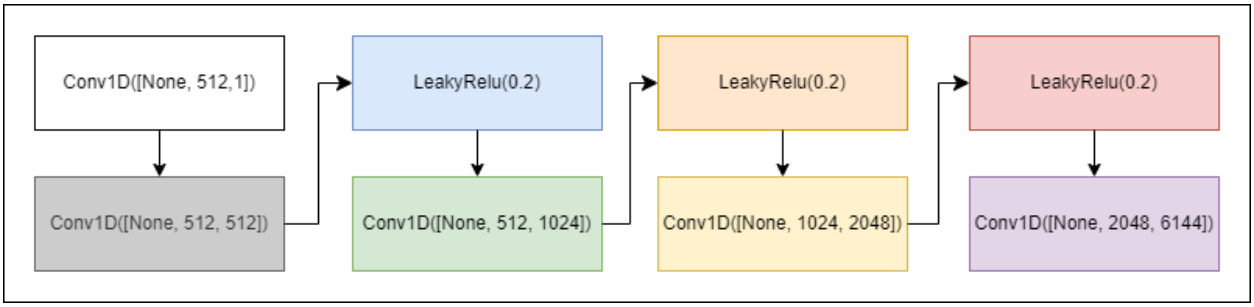


Figure 3: Generator Layers

The discriminator receives a point cloud with dimensions of 3 x 2048, which is then subjected to a series of 1D convolutions with a kernel size of one and no distortion. The outputs are subjected to the LeakyRelu activation function max(0.2x,x) between each convolution. With each convolution layer, the number of filters increases from 3 to 1024.

After the convolutional layers, the output is subjected to a MaxPool1D layer with a pool size of 1024. This is followed by three linear layers with 512, 256, and 1 neurons at the output, respectively. The goal of these linear layers is to minimize the dimensional of the output until we have a single scalar number that indicates the probability that the input is true or false.

The final output of the discriminator is a single scalar value that indicates the probability that the input point cloud is real and belongs to the training data or fake generated by the generator. If the discriminator has been trained correctly, it should give a low value for real point clouds and a high value for unrealistic point clouds.

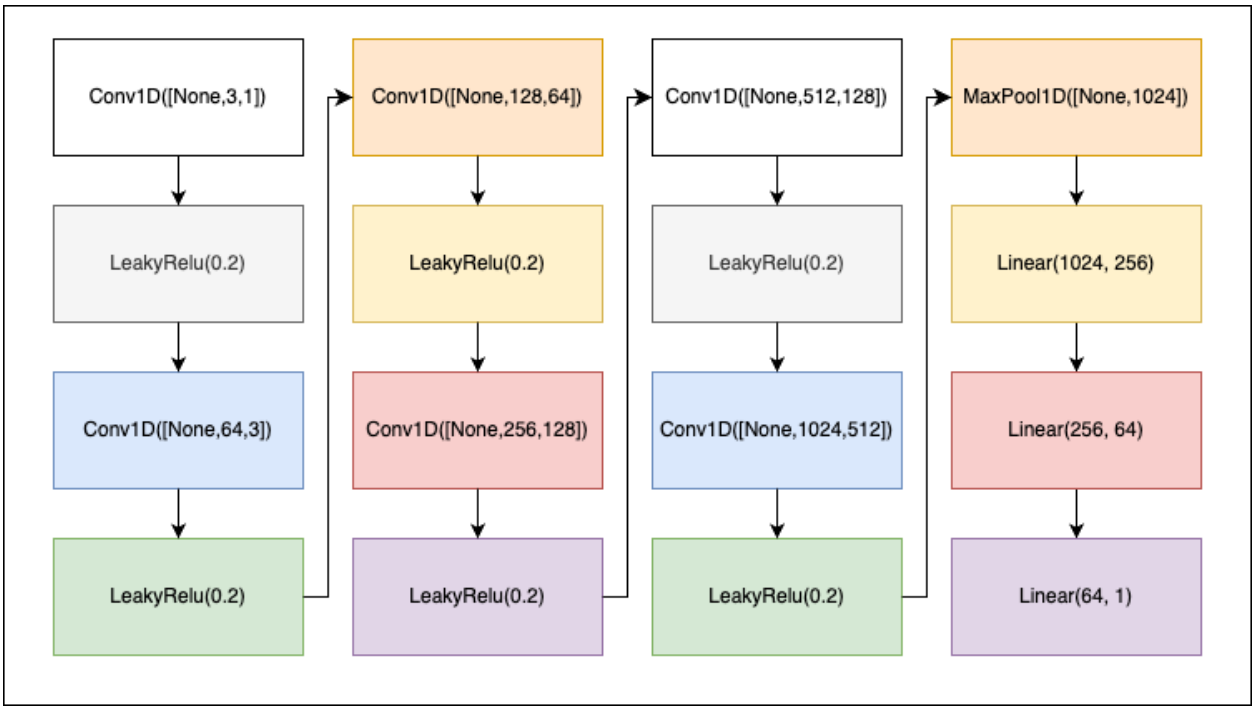


Figure 4: Generator Layers

Wasserstein loss is a type of loss function used in Generative Adversarial Networks (GANs). Unlike standard GANs that use binary cross entropy loss, Wasserstein loss evaluates the distance between the distribution of the generated data and the distribution of the original data. The Wasserstein distance, commonly known as Earth Mover's Distance, is used to calculate this distance [6].

The Wasserstein loss has the advantage of providing a smoother training process compared to other GAN losses. It also improves convergence and can lead to higher quality results over time [6].

The authors of the original Wasserstein research GAN advocated the addition of a gradient penalty term to the discriminator loss to further increase the stability of the GAN training process. This term penalizes the magnitude of the discriminator's gradients relative to its inputs. The gradient penalty prevents the discriminator from over-fitting to the training data and becoming too strong [6].

The first term (-D(x)) in the discriminator loss equation you provide reflects the output of the discriminator for an actual data sample x. The second term (-D(G(z))) reflects the output of the discriminator for a generated data sample G(z), where z is a random noise vector. The gradient penalty term (GP (D, x, G(z)) is derived based on the outputs of the discriminator for actual and generated data samples.[6].

$$L_D = -D(x) - D(G(z)) + GP(D, x, G(z))$$

Going deeper, let's see the gradient penalty together, it's objective is to improve the stability of training. [6]

$$GP = (\|\nabla_{\mathbf{x}} D(x)\| - 1)^2 * 10$$

The equation for the generator loss on the other hand is much simpler as it corresponds to the additive inverse of the output of the discriminator for the generated data sample (-D(G(z)). The purpose of the generator is to reduce this loss function, which motivates it to generate data that the discriminator is more likely to recognize as genuine. [6]

$$L_G = -(D(G(z)))$$

After generating the point cloud, using Open 3D and the ball pivoting algorithm, a mesh is constructed using some smoothing iterations, to which, in collaboration to Polyhive AI a texture is rendered. [1] [11] [2]

The code implementation is found on GitHub.

4 Ablation study

The ablation study is fully going to focus on the discriminator as removing / adding layers to the generator is not going to improve the generations as in some cases it the network isn't capable of creating realistic looking fake samples.

If we where to cut the number of liner layers to only 1, the results are going to be very unrealistic and not accurate at all, however if we would to expand it up to 6 layers the generator becomes so good at only generating models from the dataset causing it to over fit. Below there are 2 images that represent each generation with the different fully connected layers, their purpose is to bring down a high resolution array of size 1024 to only 1, by doing so it is very important to not have to many of them or to few of them, this is also influences the output generation as shown below.

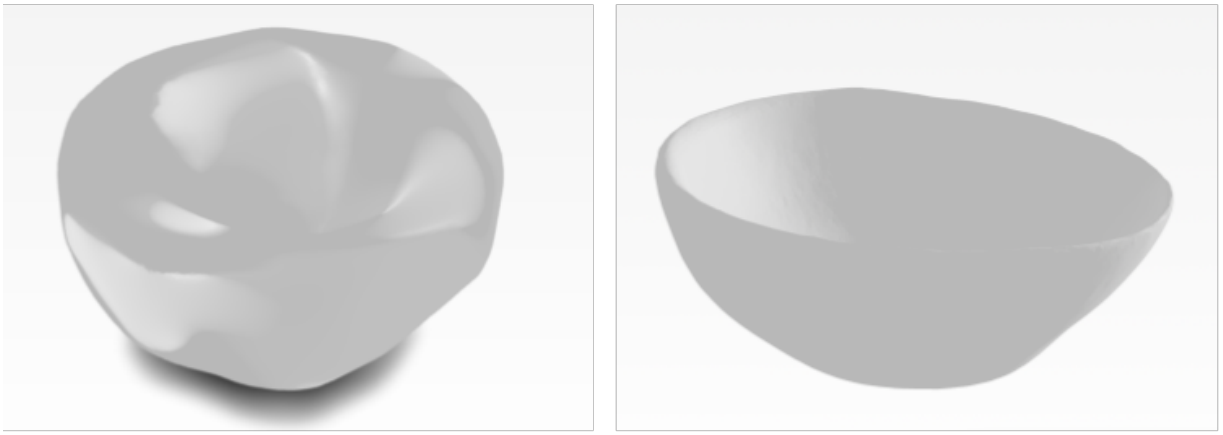


Figure 5: 1 vs 3 fully connected layers

Furthermore, another point on which I am going to focus on, is the sentence embedding, as, before passing it through the pre-trained model, natural language processing is applied. In fact if NLP is not used in order to have a proper

generation it is necessary to almost giving the same prompt as the training one. This is due to the low size of the sentence embedding which is only 512, in fact an "a" or "the" and so on can change the vector that much that the GAN struggles to generate realistic models. Below there are to images representing the effect that NLP has one the generation, on the left side it is used while on the right it isn't, it is important to outline that the prompt is the same for both pre-trained models.

Figure 6: NLP usage

As mentioned in the proposed method the loss function is the Wasserstein with Gradient Penalty, the reason behind this is the stability and the quality while training. In fact if I would you see the traditional loss implemented in the original paper [5], as the functions highly depends on hyper parameters. This also means that from a theoretical point of view also that loss could be used, however finding them is really time consuming and doesn't really payout as the generations are never going to be with the same quality. In fact by using the optimal hyper parameters for both losses, the Wasserstein one and the Vanilla one (as shown in the original paper) [5] the results are completely different as shown below.

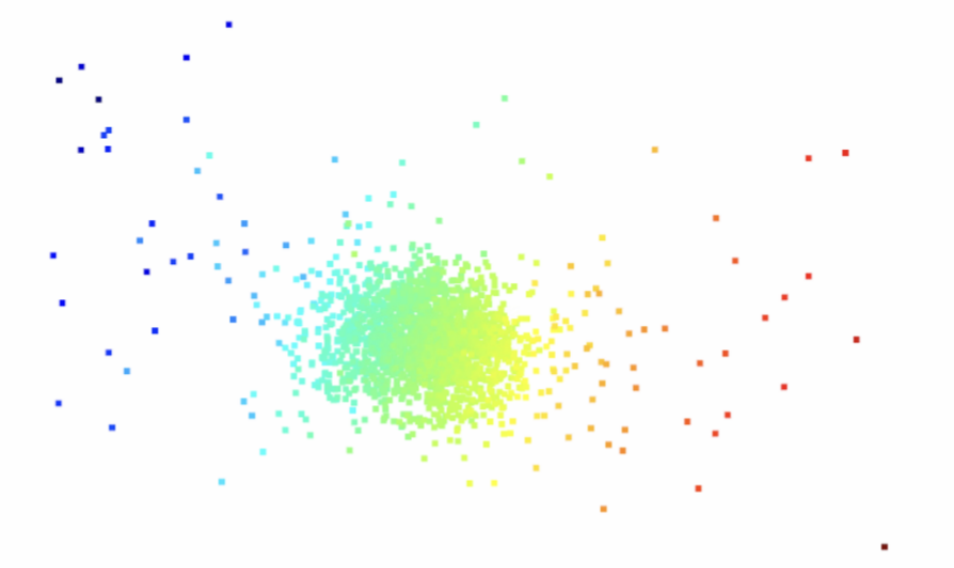


Figure 7: Vanilla GAN loss

5 Results

As mentioned earlier, Nervision uses the Wasserstein Loss with a Gradient Penalty (WGAN-GP) [6] for training the GAN. The hyper parameters used for training are as follows:

Batch Size	Epochs	LR	Beta 1	Beta 2	Optimizer
1	1000	0.0002	0	0.99	Adam

Table 1: Hyper parameters table

The learning rate, the two betas and the optimizer are exactly the ones presented in the Wasserstein GAN paper [6], while the batch size is one as each model needs to be trained singularly.

Furthermore, at first I tried to put the epochs up to 2000 however after doing some training it turned out that they were unnecessary as there were little to no differences in quality from the epoch 1000 to the 2000.

The time taken to train is very fast as it only takes 0.08 seconds for each iteration, which is about 40 percent faster than the Warping GAN model [10]. After doing some changes and messing around with different hyper parameters these are the losses:

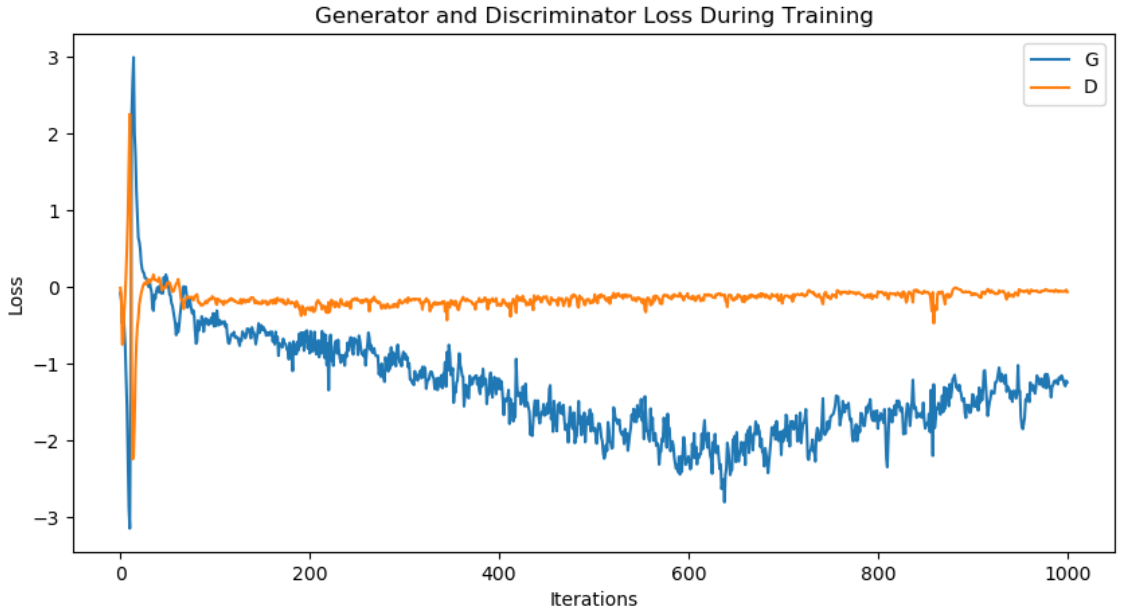


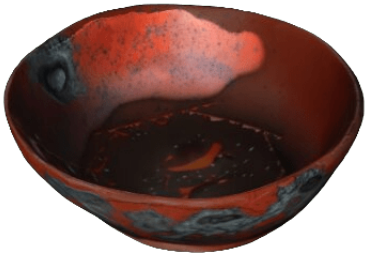
Figure 8: Generator and Discriminator Losses

Once the generator model is trained and exported it takes about 1 second to generate a model from a text, the most time consuming part is the texture applying done by Polyhive AI [1] which has a duration of about 5 minutes. Below there are a couple of distinct 3D models that have been generated by an artificial intelligence using different prompts.

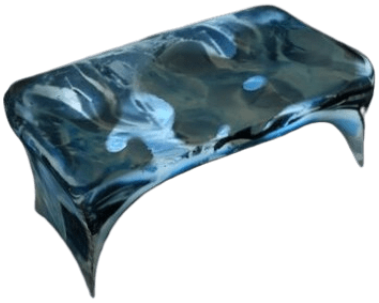
Below there are 3 different models: a bowl, a bench and a bathtub, each of them was generated by Nervision and Poly Hive AI from a simple prompt list below in the table. This shows that this GAN has a huge potential and maybe being one of the first text to 3D model AI. The dataset used for the training is very small as it is made up by only 5k models, if we were to have the same size of DALL-E [8] which is 250M it could generate very high resolution models with a vast range.

(a) Flower pattern bowl

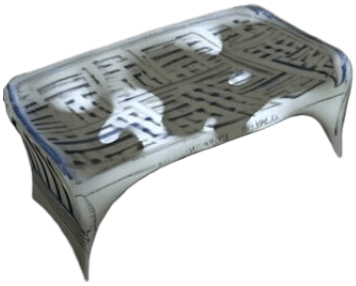
(b) Golden bowl



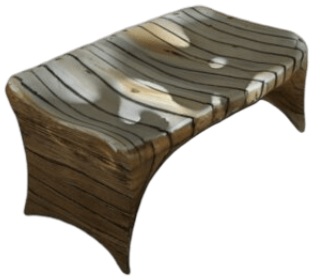
(c) Bowl made up with lava



(d) Ice bench



(e) Greek bench



(f) Bench made of wood

(g) Rainbow bathtub 4k

(h) Diamond bathtub

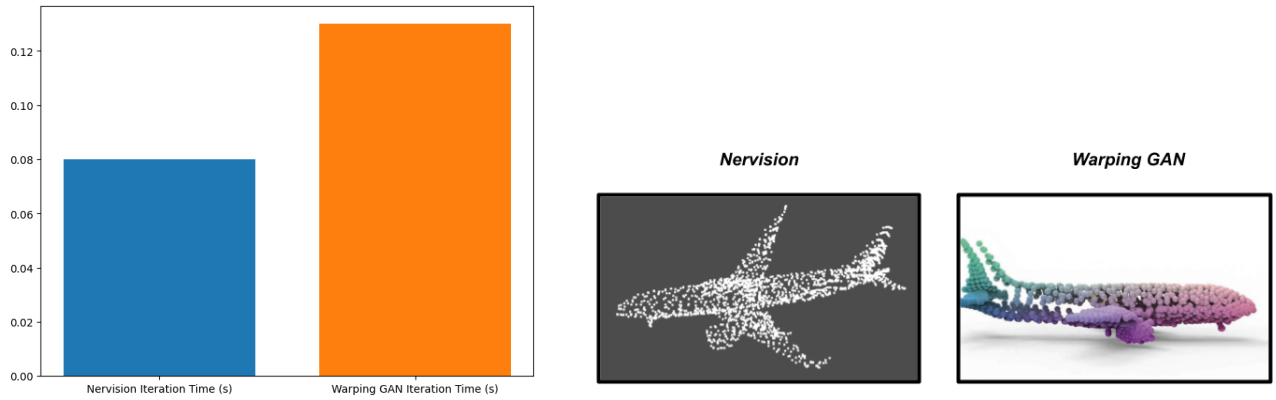
(i) Radioactive styled bathtub

6 Comparisons

The biggest benefit from using Nervision's GAN is the training time as it is really fast. When I was training it with my RTX 3090 with 24 GB of VRAM it took only 0.08 while when I was testing Warping GAN [10] it took 0.13 seconds for each iteration.

The main reason of this is because my GAN doesn't build around a grid as the one presented by Warping GAN, furthermore when passing the input z there is no Code Enhancement, however the main reason is that my loss functions is the Wasserstein GAN GP, while the Warping GAN uses the a stitching loss. [10]

Not only Nervision is about 38 percent faster, but it is also as efficient as its generations have more or less the same quality. In fact when trained on the same exact dataset the results are very similar and both are both high quality.



(a) Time comparison

(b) Quality comparison

Furthermore the point cloud generation together with the reconstruction algorithm creates a very high quality 3D model, which out stands from the Tree GAN [9] and Warping GAN [10] ones.

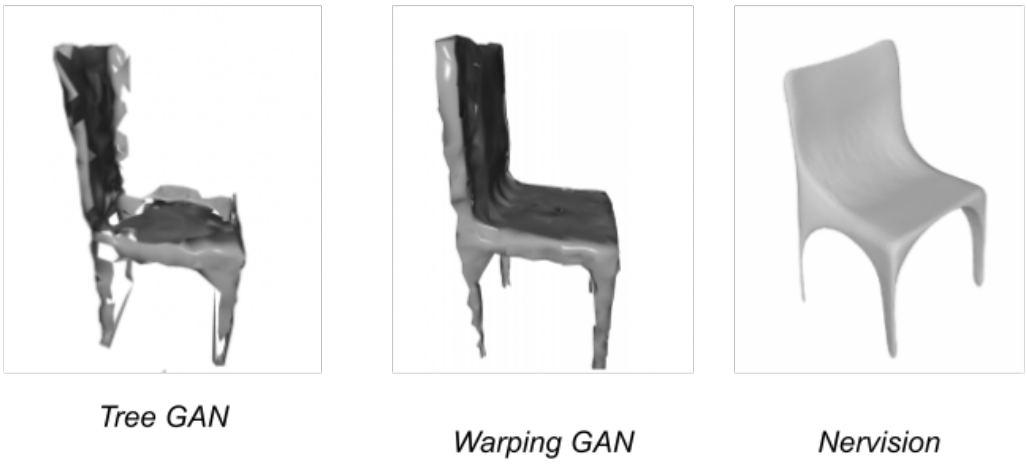


Figure 9: Model reconstruction of a chair [10]

7 Conclusion

The collaboration between Nervision and Polyhive AI could result into a breakthrough technology that is able to generate 3D models from a single text offering a lot of disadvantages.

Another important benefit of this technology is the ability to save costs and time, for example designers and game developers who are on low cost and quickly need 3D models could use Nervision to generate their assets and complete their projects in less time, which lowers the overall cost of production.

However, this technology also has some disadvantages. The accuracy and quality of the 3D models created is highly dependent on the data acquisition and training techniques used to create the AI model. Therefore, in some cases, such as when working with unusual objects it could happen that the generations don't look really realistic at all.

To furthermore improve this technology a larger and more diverse data set can be used to train the AI model. In fact the AI model can learn to create more accurate and diverse 3D models and way more powerful hardware could lead to better overall results. In addition, by improving the methods and technologies used to train and run the AI model, time savings can be realized, allowing for faster and more efficient creation of 3D models.

Finally, Nervision and Polyhive AI collaborated to create a game-changing technology capable of constructing 3D models from a single word. Although this technology has major limitations and obstacles, the potential benefits are tremendous and can lead to enhanced efficiency and cost savings for designers and developers. This technique has the ability to improve and expand in the future by using a larger dataset and better training algorithms and hardware.

References

[1] Polyhive, ai auto-texturing tool. generative ai software, 2023.

[2] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[3] Daniel Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder, 2018.

[4] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[6] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[7] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-e: A system for generating 3d point clouds from complex prompts, 2022.

[8] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021.

[9] Dong Wook Shu, Sung Woo Park, and Junseok Kwon. 3d point cloud generative adversarial network based on tree structured graph convolutions, 2019.

[10] Yingzhi Tang, Yue Qian, Qijian Zhang, Yiming Zeng, Junhui Hou, and Xuefei Zhe. Warpinggan: Warping multiple uniform priors for adversarial 3d point cloud generation, 2022.

[11] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.