UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

# CS/IT Honours Project Final Paper 2022

Title: The Use of Progressive Generative Adversarial Networks for 3D Face Generation

Author: Sebastian Oliver

Project Abbreviation: 3D-GAN

Supervisor(s): Prof. Deshen Moodley

| **Category** | *Min* | *Max* | Chosen |
|---|---|---|---|
| Requirement Analysis and Design | *0* | *20* | 0 |
| Theoretical Analysis | *0* | *25* | 5 |
| Experiment Design and Execution | *0* | *20* | 10 |
| System Development and Implementation | *0* | *20* | 10 |
| Results, Findings and Conclusions | *10* | *20* | 20 |
| Aim Formulation and Background Work | *10* | *15* | 15 |
| Quality of Paper Writing and Presentation | *10* | | 10 |
| Quality of Deliverables | *10* | | 10 |
| Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*) | *0* | *10* | 0 |
| **Total marks** | | **80** | **80** |

# The Use of Progressive Generative Adversarial Networks for 3D Face Generation

Sebastian Oliver
University of Cape Town
Cape Town, South Africa
olvseb001@myuct.ac.za

## ABSTRACT

Human faces are a lot more complex than ordinary objects, as each face is unique. This would seem to require a particularly creative approach when attempting to generate a 3D model of one. This is where Generative Adversarial Networks(GANs) come in. They have already had great success in terms of 2D face generation with some of the best results coming from progressive GANs. In this paper, we look at if a progressive GAN will have the same success in 3D face generation. Specifically, we look at 3D face generation on point cloud data as opposed to meshes which have been done in the past. We take a look at what research has been done in the field of 3D face generation and progressive GANs. Next, we introduce the 3D Progressive GAN(3D-ProGAN) and its design. In spite of the 3D-ProGAN not being able to produce state-of-the-art results, it proves that progressive GANs with just a single progressive layer are not able to capture the finer details of a complex object. This leaves the door open for a progressive GAN with multiple progressive layers that could be able to capture all the finer details due to its higher resolution.

## 1 INTRODUCTION

GANs have had great success in the past for 2D image generation. One of the main reasons is their ability to be able to be so creative with their output considering all they take as input to their generator is noise [7]. Most GANs also fall under unsupervised learning, meaning they do not require labelled training data, which makes finding usable datasets a bit easier. Taking in all these factors it made us wonder why GANs were so successful at generating 2D images but have seemed to fall short during 3D generation.
Face generation in machine learning involves training a model using facial data in the hope of generating a face, which tends to depend on both the quality and quantity of this dataset, as well as the type of model used as some models are a lot more successful when it comes to faces. Face generation using GANs has been particularly successful, especially in the 2D realm, one of the most notable being that of StyleGAN3[1] by Karras et al. [11]. When it comes to 3D face generation, one study stands out from the rest which is MeshGAN by Cheng et al. [5] which operates directly on meshes. This got us thinking that if a GAN is able to successfully produce a 3D facial model operating directly on meshes, would it be able to do the same say if the data was in the form of a point cloud. The following study will be looking at whether progressive GANs are able to generate high-fidelity 3D faces.

[1]StyleGAN3 makes use of progressive growth.

## 2 BACKGROUND AND RELATED WORK

### 2.1 Generative Adversarial Networks

GANs were first introduced by Goodfellow et al. in [7] in 2014, making them a relatively new type of machine learning. The architecture of a GAN consists of two neural networks that compete against each other. These networks are known as the generator and discriminator, whose jobs are to generate realistic outputs from noise and predict whether an input is realistic or fake. This can be seen in Figure 1.

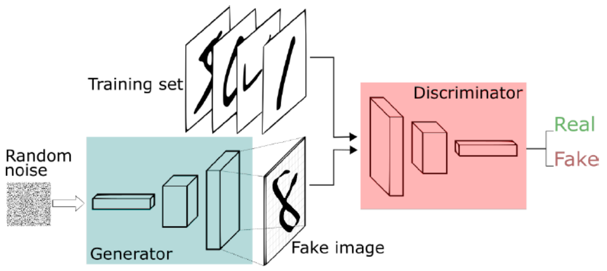

**Figure 1: Diagram of a GANs structure [2]**

In order for these two networks to compete against each other, there needs to be a way to evaluate their outputs, this is done through the use of a loss function. The loss function used in the original GAN is the minimax loss function[7]. Where the generator is looking to minimize their loss and the discriminator is looking to maximize their loss, this equation can be seen in Equation 1. There are three main categories of GAN architectures: controllable, conditional and progressive. Controllable and conditional GANs involve labelled training data that help the model learn expressions when generating its own output. Progressive GANs are all about progressively growing both the discriminator and generator simultaneously during training towards a high-resolution, which can be seen in Figure 2.

$$L_{adv} = \min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [1 - \log D(G(z))] \tag{1}$$

One of the first extensions of the original GAN is the Deep-Convolutional GAN(DC-GAN) by Radford et al. [13]. The main addition was the use of 2D convolution transpose layers in the generator and 2D convolution layers in the discriminator. One of the main takeaways from Radford et al. was that these convolutional layers proved to be a more stable architecture when training GANs[13].

## 2.2 Progressive GANs

Progressive GANs(ProGANs) was first introduced by Karras et al. in [10]. ProGANs learn concepts by first mastering them at a low level before moving onto a higher level, it can be thought about the same way people would learn a subject such as math. This progressive growth is said to bring about advantages such as faster train times as you are able to do the majority of the training at lower resolutions. The resolution is increased just before the layer converges, this helps keep the amount of new information that the generator and discriminator have to learn to a minimum. A key part of the progressive layers is the interpolation of data being passed from one layer to the next. In this original paper, additional techniques were also used between each layer. For the generator, this involves pixelwise feature vector normalization and for the discriminator, it involves minibatch standard deviation. Both of these aid in avoiding errors such as mode collapse and vanishing gradients. Another clear difference in this type of GAN is the use of an equalised learning rate which ensures both the generator and discriminator are learning at the same rate. The original ProGAN uses the Wasserstein Gradient Penalty(WGAN-GP) as its loss function, which was derived from the Wasserstein loss function first introduced by Arjovsky et al. in [1]. The equation for WGAN-GP can be seen in Equation 2.

$$\begin{aligned} L_{WGAN-GP} =& \mathbb{E}_{x_g \sim \mathbb{P}_g}[D(x_g)] - \mathbb{E}_{x_r \sim \mathbb{P}_r}[D(x_r)] \\ &+ \lambda \mathbb{E}_{x_g \sim \mathbb{P}_{xg}}[(\| \nabla_{x_g} D(x_g) \|_2 - 1)^2] \end{aligned} \tag{2}$$

This original ProGAN's aim was to generate 2D images of faces at a high resolution and they managed to outperform all other techniques at the time in this area. Karras et al. also trained their model on a bunch of other pictures like landscapes, furniture etc and got just as good results as the faces. 3D applications of ProGANs have been in the realm of brain volumes and object generation, with 3D face generation having not been explored yet according to our research.

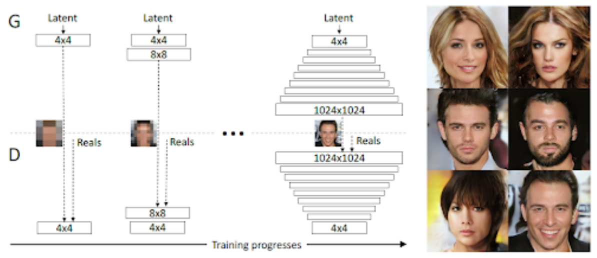

**Figure 2: Overview of progressive training [10]**

## 2.3 3D Face Generation

Face generation is one of the many different facial applications that GANs have been applied to. These types of GANs have evolved over time from uncontrollable grey-scale images to photo-realistic, high-resolution images with feature control. So far the majority of research has been done in the realm of 2D face generation, including techniques such as image-super resolution, image-to-image translation, image matting, face completion and face reenactment. This leaves the door open for 3D face generation.
When working with a 3D model such as faces, care needs to be taken in which technique you are going to use to represent these 3D faces. Modelling techniques include:

*2.3.1 3D morphable models.* Based on two main ideas, the first being that all faces are in the form of point-to-point correspondence and the second is to separate facial colour and shape.

*2.3.2 Meshes.* Involves dividing up an image into connecting polygons, the most commonly used one being the 3D triangle mesh. Creating and working with meshes is usually computationally expensive.

*2.3.3 Point clouds.* Are an unordered set of 3D points that may have an additional bit of information representing the RGB of each point.

*2.3.4 Voxels.* Can be thought of as 3D pixels, where each pixel is represented as a 3D cube and a collection of voxels come together to form a 3D model.

3D face generation can be split into two types: 3D face generation from 2D data and 3D face generation from 3D data. Our interest is in the latter, hence the explanation of the different types of 3D modelling techniques.

## 2.4 CoMA

A convolutional mesh autoencoder that learns a non-linear 3D representation of a face[14]. The main reason for Ranjan et al. for choosing to learn a non-linear representation is so that they are able to learn extreme deformations and non-linear expressions, something traditional models have not been able to achieve since they tend to learn a linear representations[14]. Autoencoders differ from traditional GANs as they have down-sampling and up-sampling layers between Chebyshev convolutional layers in their architecture. CoMA's architecture can be seen as an inverse to a ProGAN, which can be seen architecture-wise in Figure 3.

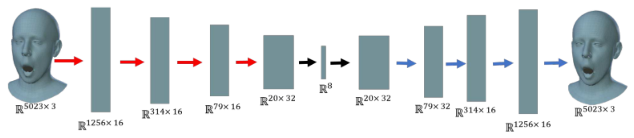
$\mathbb{R}^{5023\times 3}$
$\mathbb{R}^{1256\times 16}$
$\mathbb{R}^{314\times 16}$
$\mathbb{R}^{79\times 16}$
$\mathbb{R}^{20\times 32}$
$\mathbb{R}^{8}$
$\mathbb{R}^{20\times 32}$
$\mathbb{R}^{79\times 32}$
$\mathbb{R}^{314\times 16}$
$\mathbb{R}^{1256\times 16}$
$\mathbb{R}^{5023\times 3}$

**Figure 3: Diagram showing the layers of CoMA, red arrows indicate down-sampling and blue indicates up-sampling[14]**

## 2.5 MeshGAN

MeshGAN is able to generate 3D facial images of different identities and expressions. Cheng et al. were able to do this by implementing the first GAN architecture to generate 3D meshes using convolutions directly on 3D meshes [5]. The reason is that previous research has been on GANs operating directly on volumetric representations of 3D objects, of which they have always seemed to fall short. This architecture was based on the BEGAN architecture implemented in [3]. Like CoMA above, MeshGAN also makes use of Chebyshev convolutional layers in their architecture.

## 3 DESIGN AND IMPLEMENTATION

### 3.1 Dataset

The CoMA triangle mesh dataset, first presented in [14] by Ranjan et al., is what we have chosen to work with. One of the main reasons is that MeshGAN is our reference study. It consists of 20 466 3D meshes, each of which contains about 120 000 vertices as specified in [14]. The dataset has 12 different subjects each with 12 different classes of extreme expressions.

### 3.2 Preprocessing

Since our dataset is the CoMA triangle mesh dataset, it made sense to make use of the same preprocessing that was used in [14]. This meant using an implementation of a sequential mesh registration method by Li et al. in [12] to interpolate the data down to 5023 vertices. Additional down-sampling needed to be done in order to train the progressive GAN. This was done using Open3D and NumPy functions, which down-sampled the data to a further 512 and 1024 vertices.

### 3.3 Metrics

The choice of metrics was based on what's most relevant in the area of face generation and GANs, specifically from [9]. Another big factor in what metrics were chosen was based on the metrics that were used by Cheng et al. in [5], due to MeshGAN being our reference study. The FID and KID calculations were done using Abdul Fatir's open source implementation found here [6].

*3.3.1 Generalisation.* Measures the model's ability to generate unseen face shapes that were not part of the training set. It is calculated using a per-vertex Euclidean distance between each sample in the test set and the corresponding generated sample. Then we take the average over all the vertices. This is the same method they use to calculate generalisation in MeshGAN [5].

*3.3.2 Specificity.* Looks at the validity of the generated faces. This is done by finding the generated face's nearest neighbour in the test set using the minimum of the average per-vertex distance [5].

*3.3.3 Frechet Inception Distance(FID).* First introduced by Heusel et al. in [8], it measures the quantity and diversity of the generated samples. It calculates how close the generated sample is to a real one by looking at its probability distributions. This is done by using a pre-trained network called the Inception Network. But this comes with a problem as the Inception Network is trained on images, therefore the 3D generated output had to be rendered into an image. This was done using the visualisation framework mentioned above.

*3.3.4 Kernel Inception Distance(KID).* Introduced by Binkowsji et al. in [4], this metric is very similar to FID above, the only real difference is the type of distance used to measure between the generated sample and the real one. It also makes use of a pre-trained Inception Network.

### 3.4 Experimental Platform

The experimental platform consists of two different frameworks, which both come together to be able to easily interchange a type of GAN architecture for training and evaluation purposes. This platform was worked on together as a team with members having specific focuses[2].

*3.4.1 Training Framework.* The main functions of the training framework are the following. Loading the preprocessing data and splitting it into training and testing data. It then takes this training data and uses it to train the model specified. During training, the framework saves the model and output of this model every couple of epochs. Figure 4 shows what the architecture of the framework looks like.

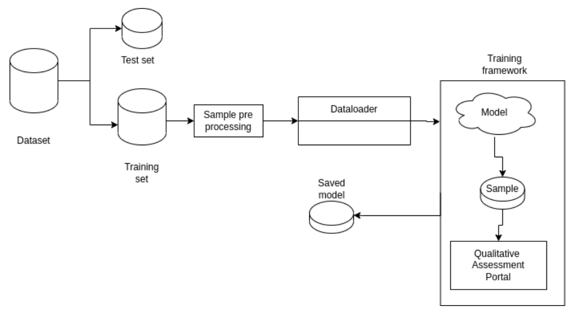

**Figure 4: Diagram showing an overview of the training framework**

*3.4.2 Testing Framework.* The testing framework's main function is to evaluate the model in question. This is done through a series of metrics. The metrics include KID, FID, generalisation and specificity. These metrics are then outputted at the end of execution. A diagram of the architecture used for this framework can be seen in Figure 4.

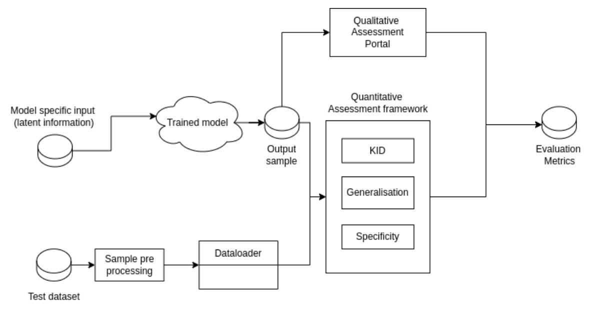

**Figure 5: Diagram showing an overview of the testing framework**

### 3.5 Visualisation

As the output of our networks will be 3D models of faces of some form, a visual representation of these is needed to ensure that what is being produced is actually viable as a face and that the models have not just learnt how to perform well according to the metrics. This brought about a need for a visualisation framework[3]. The framework was developed with two main viewing options in

[2]Sebastian Oliver implemented the base training framework, Luc van den Handel implemented the testing framework and Liam Watson designed both the training and testing framework diagrams.

[3]Liam Watson implemented this visualisation framework

mind. One is a raw point cloud that is easy to view during training to ensure the generator is not converging to issues such as mode collapse or vanishing gradients. The other viewing option is that of a meshed object which is done using Poisson Surface Reconstruction on the outputted point cloud from the model.

## 3.6 Simple-DCGAN [4]

This GAN was based on the DCGAN by Raford et al. in [13] with the aim of providing a base implementation upon which our other GANs can be built upon. The main difference between the original DCGAN and our simple-DCGAN is the type of convolution layers and convolution transpose layers that are used for the discriminator and generator. Radford et al. made use of 2D convolution and convolution transpose layers while ours uses 1D convolution and convolution transpose layers. The DCGAN uses 2D layers as they were aiming to generate 2D images. Since our aim is to generate a point cloud with 3D coordinates, it made sense to use 1D layers so our coordinates are stored in a 1D array with each element having a depth of 3. The 3 values of an element represent an x, y and z coordinate. The loss function used for the Simple-DCGAN is the same as that used in the DCGAN [13], which is the minimax loss and can be seen in Equation 1.

*3.6.1 Generator.* The generator network consists of 6 1D convolution transpose layers, each separated by a ReLU activation function. An overview of these layers can be seen in Figure 6. All layers except the last layer have a kernel size of 1, the last layer has a kernel size of 3 as this is required to get the output into the correct format of a 1D array with a depth of 3. The final output is then passed through a Tanh activation function to ensure its a value between -1 and 1.

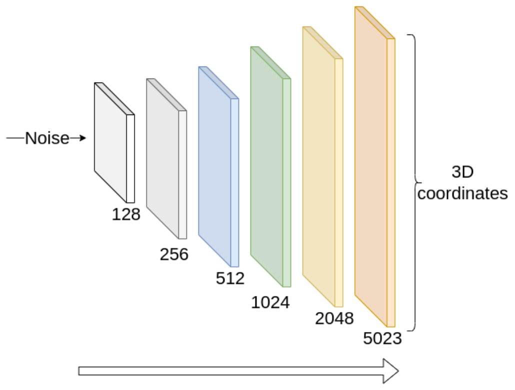

**Figure 6: Diagram showing the Simple-DCGAN's generator's layers**

*3.6.2 Discriminator.* The discriminator network consists of 7 1D convolution layers, each separated by a leaky ReLU activation function(another difference compared to that of the generator). The different sizes of each layer can be seen in Figure 7. All layers have a kernel size of 1 as the input passed in is already in the correct format. The final layer is passed through a sigmoid activation function, which ensures the output is between 0 and 1. This represents whether the discriminator thinks the passed input is real or fake. The closer to 1 it is, the more real it thinks it is.

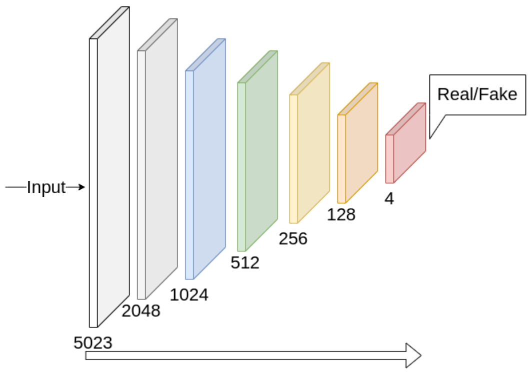

**Figure 7: Diagram showing the Simple-DCGAN's discriminator's layers**

## 3.7 3D-ProGAN

As mentioned above, this GAN is based on the Simple-DCGAN as well as the original ProGAN by Karras et al. in [10] in order to get an idea of how to implement progressive growth. The main addition to the Simple-DCGAN is batch normalisation, up-sampling and skip-connections that happen between layers in the generator. The idea behind the design was to start out with the Simple-DCGAN without the last three layers in the generator and without the first three layers in the discriminator and then progressively grow both networks during training. In order to ensure both networks learn at the same rate, they needed to both grow simultaneously otherwise issues such as vanishing gradients could occur and result in incorrect output. In this study, implementing the WGAN-GP on 1D convolutional layers proved difficult, therefore it was chosen to stick with minimax loss, which can be seen in Equation 1. An important design choice is that there will be only a single progressive layer included in its architecture, with the idea being that implementing multiple layers would just require using the same method used for a single layer multiple times.

*3.7.1 Generator.* The core of the generator consists of three base layers, which can be seen in Figure 8 and which consist of 1D transpose convolution layers separated by a 1D batch normalisation layer and a ReLU activation function. The batch normalisation layer was introduced to prevent the vanishing gradient problem which occurred after the addition of the first progressive layer. This core network outputs point clouds with 512 points. To increase the size of this point cloud progressively training the generator needs to occur which involves two different methods, both of which were used in the original ProGAN[10]. First, the output of the previous layer is up-sampled to twice the size. Next, this up-sampled point cloud is then parsed through a 1D transpose convolution layer as well as skipping this layer(this is the skip connection mentioned earlier). This skip connection is then merged with the output from the transpose convolution layer and passed to a 1D batch normalisation layer and ReLU activation function. The resulting point cloud is then output.

[4]Liam Watson implemented this GAN

*3.7.2 Discriminator.* The core of the discriminator consists of three base layers and a single output layer. The base layers consist of 1D convolution layers separated by a Leaky ReLU activation function, while the output layers consist of a 1D convolution layer with a Sigmoid activation function. In order to progressively grow the discriminator, no down-sampling, batch normalisation or skip connections were needed. These three different techniques were tested in the discriminator but these always led to issues such as a vanishing gradient problem. The structure is very similar to that of the Simple-DCGAN's structure, the only difference being that at the start of training it only has 4 layers as opposed to the full 7 layers. It is grown to 5 layers during its progressive growth.

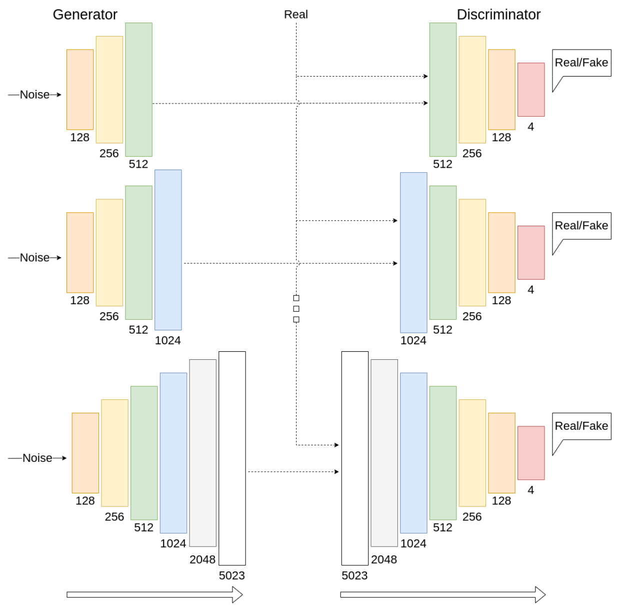

**Figure 8: Diagram showing the 3D-ProGAN's progressive growth during training.**

## 3.8 Hyper-parameter Tuning

The following hyper-parameters in Table 1 were used for training both the generator and discriminator of the 3D-ProGAN. A vanishing gradient problem seemed to keep on occurring when training the 3D-ProGAN and these parameters helped avoid this. In Table 1, a scheduler is a way of decaying the learning rate during training. The type of scheduler used for both the generator and discriminator is the StepLR, that decays the learning every 15 epochs during training by 0.25 of the previous learning rate.

**Table 1: Table showing the different hyper-parameters used for training**

| Hyper-parameter | Value |
|---|---|
| Loss function | BCE loss |
| Optimizer | Adam |
| Learning rate | 1e-5 |
| StepLR scheduler | 15 |
| Batch size | 10 |
| Noise dimension | 128 |
| Epochs | 1000 |
| Progressive layers | 1 |

## 3.9 Implementation

The decision was made to use Python for this project, one of the main reasons being that it offers the greatest variety of machine learning libraries. The following python libraries were used:

- Pytorch, as the machine learning library
- Matplotlib, for visualizing the results during training
- NumPy, to manage our input arrays
- Open3D, for 3D visualization and down-sampling
- tqdm, for a progress bar during training
- random, for generating a random seed upon initialization

## 4 RESULTS AND DISCUSSION

In both Tables, 2 and 3 generalisation and specificity are shown in a standard deviation which is calculated over all the faces generated by the models' trained generators. While for FID and KID, it is a single value that is the calculated mean value on all the models' generated faces. When interpreting the point cloud representations of the generated samples, please keep in mind the colour key represented in Figure 9 where it represents the value of the z-coordinate in the point cloud.

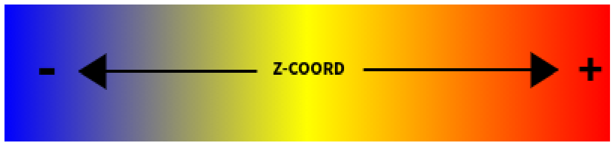

**Figure 9: Diagram showing colour key used in the point cloud visualisation.**

## 4.1 Simple-DCGAN

*4.1.1 Visualisation.* In Figure 10 we are able to see the output of the same face from different angles and in different representations from the Simple-DCGAN's generator. Both of these are generated using the visualisation framework mentioned in Section 3.4. At the top of the figure, the data is represented using a point cloud. In this form, we are able to see that the GAN has learnt where every feature of a face is and is able to make out the rough shape of these features. The reconstructed mesh(the bottom half of Figure 10) does not fairly represent the GAN's ability to generate the shape of these features as part of this reconstruction, involving estimating normals which are never going to be perfect. These figures prove that the Simple-DCGAN is able to generate a 3D facial structure and is a good base to build upon.

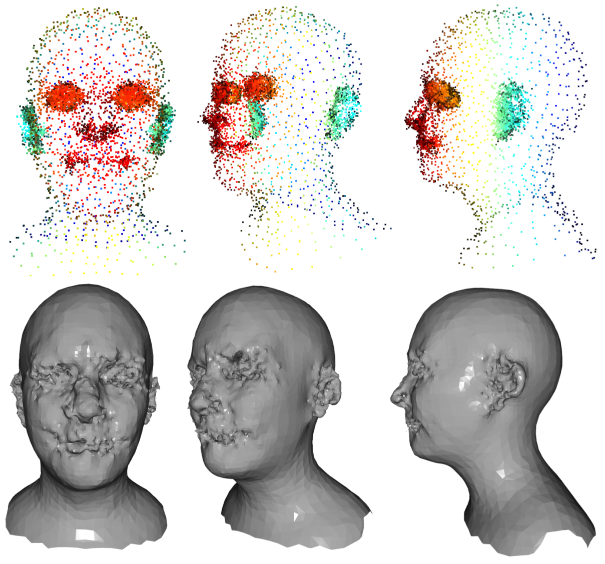

**Figure 10: Diagram showing the output of the Simple-DCGAN's generator after training in both a point cloud(on top) as well as a mesh(underneath).**

*4.1.2 Metrics.* Now we move to the quantitative evaluation. In Table 2 the metrics are shown for the Simple-DCGAN, which were calculated using the testing framework mentioned in Section 3.3.2. The four metrics mentioned in Section 3.5 were used to evaluate it from a quantitative perspective. For all the metrics, a lower value is best. Therefore the KID and FID values indicate that the Simple-DCGAN is able to generate a 3D facial model that is of high quality and very close to that of an actual face. The low generalisation value indicates the generator is able to generate unseen face shapes, while the low specificity value indicates its ability to generate a valid face, where valid means the face it generates is somewhat similar to a face in the dataset. Compare that against the randomly generated noise values, it shows just how much the Simple-DCGAN has learnt to generate a face.

**Table 2: Table showing a comparison of metrics between our models and randomly generated noise**

| | Generalisation | Specificity | KID | FID |
|---|---|---|---|---|
| Noise | 164.289±0.015 | 149.383±27.118 | 0.97 | 102.70 |
| Simple-DCGAN | 0.754±0.137 | 0.651±0.166 | 0.12 | 36.34 |
| 3D-ProGAN(0) | 12.195±0.190 | 11.619±0.327 | 0.29 | 57.52 |
| 3D-ProGAN(1) | 11.773±0.167 | 11.427±0.219 | 0.19 | 47.28 |

## 4.2 3D-ProGAN

In Tables 2, 3 and Figures 11, 12 the number in the brackets after the 3D-ProGAN indicates the number of progressive layers that it has. The (0) is zero progressive layers which is the point cloud with 512 vertices and the (1) is a single progressive layer resulting in a point cloud with 1024 vertices.

*4.2.1 Visualisation.* The 3D-ProGAN's generator with no progressive layers is able to pick up the core structure of a 3D facial model but lacks a lot in terms of detail for facial features, as can be seen in Figure ??. This is explained by the less dense point cloud that is generated, as fewer points mean requiring learning the same amount of information about an object with fewer resources. The point cloud representation is underwhelming due to fewer points and hard to get a general feel for the model that is being generated, while the mesh reconstruction serves as a better representation of what the generator understands about the face at a low resolution of 512 vertices. Adding a single progressive layer helped the generator pick up on a lot more details. This can be seen in Figure 12 where we are able to see a lot denser point cloud compared to that in Figure 11. The facial features have become a lot more prominent compared to when there were no progressive layers.

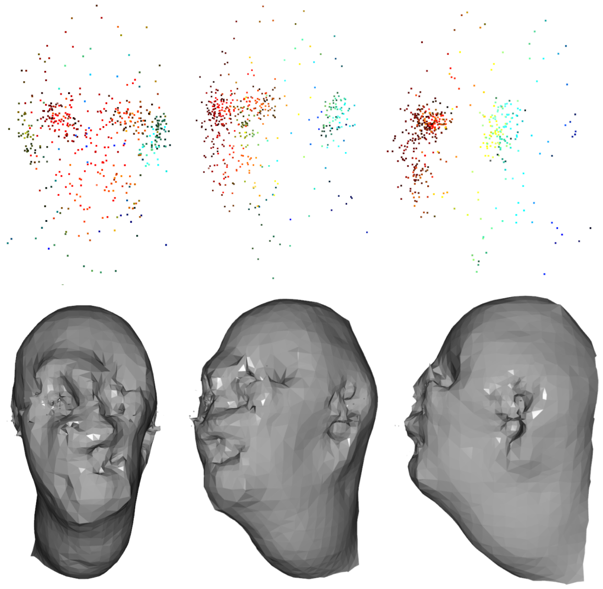

**Figure 11: Diagram showing the output of the 3D-ProGAN(0)'s generator in both a point cloud(on top) as well as a mesh(underneath).**

*4.2.2 Metrics.* In Table 2 the metrics for the ProGAN without and with a progressive layer can be seen. These metrics back up the findings when comparing Figure 11 and Figure 12 as all values get better, which in this case is lower when a progressive layer is added. The higher generalisation values for both without and with the progressive layer are higher than expected. Meaning the generator struggles with recognising unseen face shapes. The higher specificity value indicates that the faces generated do not resemble faces in the dataset too closely. The KID is under 0.3 for both which tells us that the generator is able to generate a 3D facial model that is lacking in the finer details as this represents the quality of the model generated. The FID score of around 50 for both indicates that the generated sample still has a way to go to be a realistic sample.

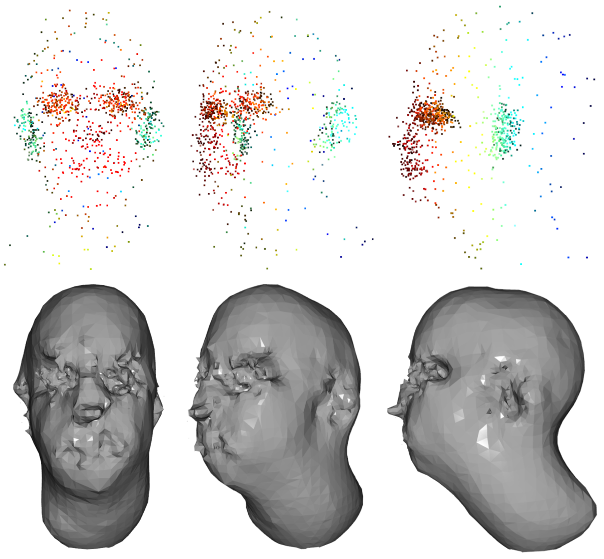

**Figure 12: Diagram showing the output of the 3D-ProGAN(1)'s generator in both a point cloud(on top) as well as a mesh(underneath).**

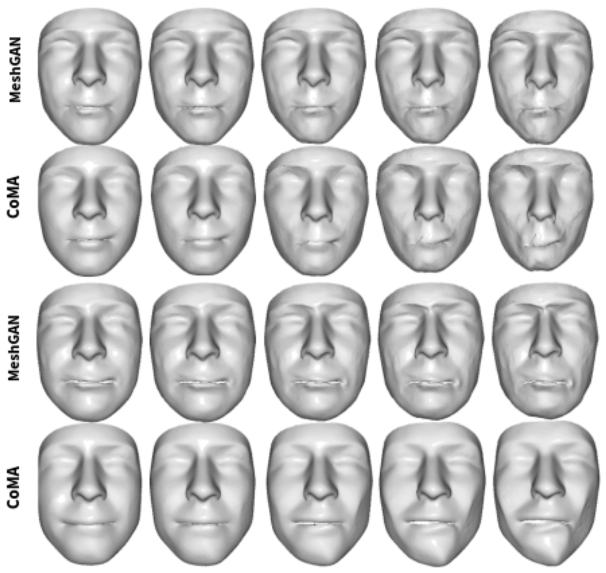

**Figure 13: Diagram showing the output of MeshGAN[5] and CoMA[14] as seen in [5].**

### 4.3 Comparison with Previous Work

Our main studies of comparison will be those introduced in Sections 2.4 and 2.5, namely MeshGAN and CoMA. Comparing Figures 13,10 and 12, we are able to see that neither the Simple-DCGAN nor the 3D-ProGAN(1) is not able to capture the same level of detail as the state of the art techniques. Although this may be due to the estimated normals during mesh reconstruction leading to less detailed features.
A comparison of the metrics can be seen in Table 3, also note there is no KID as this metric was not used in [5]. The 3D-ProGAN does not outperform MeshGAN or CoMA with regards to these metrics but the Simple-DCGAN is able to outperform MeshGAN and CoMA for specificity. Meaning it is able to generate faces that more closely resemble a face in the dataset than MeshGAN and CoMA. 3D-ProGAN's metrics are underwhelming as the closest metric(FID) is more than double that of CoMA.

**Table 3: Table showing a comparison of metrics between our work and past work**

| | Generalisation | Specificity | FID |
|---|---|---|---|
| CoMA(EXP) | 0.606±0.203 | 1.899±0.272 | 22.43 |
| MeshGAN(EXP) | 0.605±0.264 | 1.536±0.153 | 13.59 |
| Simple-DCGAN | 0.754±0.137 | 0.651±0.166 | 36.34 |
| 3D-ProGAN(1) | 11.773±0.167 | 11.427±0.219 | 47.28 |

### 4.4 Discussion

Karras et al. proved that progressive growth of a GAN resulted in a higher resolution output when it came to 2D object generation[10]. The 3D-ProGAN proved that this is the case for 3D face generation too. In the visualisation of the 3D-ProGAN, the addition of a progressive layer clearly adds more detail to the different facial features of the 3D model, while the metrics also improved when compared to that of a 3D-ProGAN without a progressive layer. Since the visualisation and metrics agree about progressive growth increasing the quality of the generated sample, it leaves the door open for how good a 3D-ProGAN could get with multiple progressive layers.
Although the 3D-ProGAN does not seem to perform any better than its core, the Simple-DCGAN, we could say a comparison between the two is not completely fair. This is due to the 3D-ProGAN having an output resolution of 1024 vertices while the Simple-DCGAN has an output resolution of 5023 vertices. Meaning the 3D-ProGAN is being judged on trying to represent a 3D facial model with less memory than the Simple-DCGAN.

## 5 CONCLUSIONS

The 3D-ProGAN was able to show that ProGANs with a single progressive layer are not able to generate usable 3D facial models. Although a single progressive layer does increase the quality of the sample generated, it does not quite keep up with the latest techniques with regard to 3D face generation. This increase in quality of the generated sample agrees with [10] in terms of progressive growth resulting in a higher quality sample. But unfortunately, the main takeaway is that the 3D-ProGAN was not able to produce high-fidelity 3D faces.

## 6 LIMITATIONS AND FUTURE WORK

### 6.1 Dataset size

As 3D data is harder to come by compared to 2D data, no one really knows how good GANs could get at generating 3D faces with enough data. For ProGANs for example, the dataset that Karras et al. used to train the original ProGAN[10] was the CelebA-HQ dataset which consists of 30 000 images, that's 10 000 more samples than the CoMA dataset. Therefore as larger and larger datasets become readily available, GANs should become better and better too as they will have more data to learn from.

### 6.2 Multiple Progressive Layers

The 3D-ProGAN evaluated above only had a single progressive layer at its highest resolution. The difference in output generated between having zero progressive layers and having a single progressive layer was apparent, meaning it should just keep on picking up on more and more of the finer details as it progressively grows more and more.

### 6.3 GAN Loss Functions

Over the past couple of years since GANs were first introduced, the biggest changes have come in the type of loss function used [9]. It has such a big impact on the GAN as it impacts how its generator and discriminator learn during training. In Section 2.2, it was mentioned that the original ProGAN uses the WGAN-GP loss function, which can be seen in Equation 2. For the 3D-ProGAN the original GAN minimax loss was used. Switching the loss function to the WGAN-GP loss function proved difficult in this application but it would surely improve the GAN as it has done to other models in the past. It was proven to eliminate issues such as vanishing gradients and mode collapse [1] which did pop up during training the 3D-ProGAN. Therefore switching to WGAN-GP could be considered as an extension to the 3D-ProGAN to improve its output.

### 6.4 Minibatch Standard Deviation and Pixelwise Feature Normalisation

Specific to that of ProGANs are the mechanisms you have in place between progressive layers in both the generator and discriminator. Karras et al. proposed minibatch standard deviation in the discriminator and pixelwise feature normalisation [10]. Minibatch standard deviation is a simplified solution of minibatch discrimination proposed by Salimans et al.[15] that helps fix the issue of GANs only learning a subset of the variation found in the training dataset[10]. Pixelwise feature normalisation controls the magnitudes in the generator and discriminator as they have a tendency of spiraling out of control due to the competition between them [10]. Karras et al. also noticed that it doesn't seem to have much of an impact on the generator's output. It is implemented between each of the convolution layers in the generator. Therefore extending the 3D-ProGAN to include minibatch standard deviation and pixelwise feature normalisation should help produce more consistent results and avoid issues such as vanishing gradients.

## ACKNOWLEDGMENTS

A special thanks to our supervisor Professor Deshen Moodley for his insight into the world of GANs and for pushing us to take a path less explored which kept us on our toes.
Secondly, a huge thanks to Luc van den Handel and Liam Watson, who form the rest of this research group and their contributions to the experimental platform and Simple-DCGAN.

## REFERENCES

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *International conference on machine learning*. PMLR, 214–223.
[2] Alan Berman. 2020. *Generative adversarial networks for fine art generation*. Master's thesis. University of Cape Town.
[3] David Berthelot, Thomas Schumm, and Luke Metz. 2017. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717* (2017).
[4] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. 2018. Demystifying mmd gans. *arXiv preprint arXiv:1801.01401* (2018).
[5] Shiyang Cheng, Michael Bronstein, Yuxiang Zhou, Irene Kotsia, Maja Pantic, and Stefanos Zafeiriou. 2019. Meshgan: Non-linear 3d morphable models of faces. *arXiv preprint arXiv:1903.10384* (2019).
[6] Abdul Fatir. 2018. FID and KID PyTorch Implementation. https://github.com/abdulfatir/gan-metrics-pytorch
[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014).
[8] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems* 30 (2017).
[9] Amina Kammoun, Rim Slama, Hedi Tabia, Tarek Ouni, and Mohmed Abid. 2022. Generative Adversarial Networks for face generation: A survey. *ACM Computing Surveys (CSUR)* (2022).
[10] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2017. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196* (2017).
[11] Tero Karras, Samuli Laine, and Timo Aila. 2019. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 4401–4410.
[12] Tianye Li, Timo Bolkart, Michael J Black, Hao Li, and Javier Romero. 2017. Learning a model of facial shape and expression from 4D scans. *ACM Trans. Graph.* 36, 6 (2017), 194–1.
[13] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).
[14] Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J. Black. 2018. Generating 3D Faces using Convolutional Mesh Autoencoders. In *Proceedings of the European Conference on Computer Vision (ECCV)*.
[15] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training gans. *Advances in neural information processing systems* 29 (2016).